Microsoft

Разработка требований к программному обеспечению

Издание третье, дополненное

Практические приемы сбора требований и управления ими при разработке программных продуктов

Карл Вигерс и Джой Битти

РУССКАЯ РЕДАКЦИЯ

bhv

Посвящается Крис.

— Крис Вигерс

Моим родителям Бобу и Джоанн.

— Джой Битти

Karl Wiegers and Joy Beatty

Software Requirements

Third Edition

Microsoft® Press

Карл Вигерс и Джой Битти

Разработка требований к программному обеспечению

Издание третье, дополненное

РУССКАЯ РЕДАКЦИЯ bhv®

2014

УДК 004.738.5
ББК 32.973.202
В41

Вигерс Карл, Битти Джой

В41 Разработка требований к программному обеспечению. 3-е изд., дополненное / Пер. с англ. — М. : Издательство «Русская редакция» ; СПб. : БХВ-Петербург, 2014. — 736 стр. : ил.

ISBN 978-5-7502-0433-5 («Русская редакция»)
ISBN 978-5-9775-3348-5 («БХВ-Петербург»)

Эта книга — подробное руководство по разработке качественных требований к программному обеспечению. Здесь описаны десятки проверенных на практике приемов выявления, формулирования, разработки, проверки, утверждения и тестирования требований, которые помогут разработчикам, менеджерам и маркетологам создать эффективное ПО. Настоящее издание дополнено новыми приемами, посвященными разработке требований в проектах гибкой разработки (agile).

Основная аудитория — бизнес-аналитики и разработчики, а также дизайнеры, программисты, тестировщики и другие члены команды, задача которых понять и удовлетворить чаяния клиентов, а также маркетологи, менеджеры по продуктам и менеджеры проекта, которые должны проникнуться «духом» и особенностями продукта, чтобы сделать его в полной мере конкурентоспособным.

Книга состоит из 32 глав, 3 приложений и словаря терминов.

УДК 004.738.5
ББК 32.973.202

Вигерс Карл, Битти Джой

Разработка требований к программному обеспечению

Издание третье, дополненное

Совместный проект издательства «Русская редакция» и издательства «БХВ-Петербург».

РУССКАЯ РЕДАКЦИЯ bhv®

Подписано в печать 21.04.2014 г. Формат 70x100/16.
Усл. физ. л. 46. Тираж экз. Заказ №

Первая Академическая типография «Наука», 199034, Санкт-Петербург, 9 линия, 12/28

Оглавление

Введение

Несмотря на более чем пятидесятилетнее существование компьютерной отрасли, многие компании-разработчики программного обеспечения по-прежнему прикладывают значительные усилия для сбора, документирования и управления требованиями к ПО. Недостаточный объем информации, поступающей от пользователей, требования, сформулированные не полностью, их кардинальное изменение и неправильно понятые бизнес-цели — вот основные причины, из-за которых зачастую командам, работающим в области информационных технологий, не удается успешно завершить проект. Многие разработчики не умеют спокойно и профессионально собирать требования пользователей к ПО. У клиентов зачастую не хватает терпения участвовать в разработке требований к ПО. Иногда участники проекта даже не могут прийти к единому мнению, что же такое «требование». Как заметил один писатель, «программисты скорее предпочтут расшифровать слова классической песни Кингсмена (Kingsmen) «Louie Louie», чем требования клиентов» (Peterson 2002).

Второе издание книги «Разработка требований к программному обеспечению» вышло за 10 лет до выхода этой книги. В мире информационных технологий это большой срок. Многое поменялось за это время, но кое-что осталось неизменным. Вот основные тенденции прошедшего десятилетия:

- признание бизнес-анализа профессиональной дисциплиной и расширение профессиональной сертификации и организаций, таких как International Institute of Business Analysis и International Requirements Engineering Board;
- достигли высокого уровня развития средства для управления требованиями в базах данных и для поддержки разработки требований, в том числе для создания прототипов, моделирования и имитации;
- распространение методов гибкой разработки (agile) и развитие приемов работы с требованиями в проектах гибкой разработки;
- активное использование визуальных моделей для представления знаний о требованиях.

Так что же *не* изменилось? Есть два обстоятельства, который делают этот раздел важным и уместным. Во-первых, во многих программах обучения разработке ПО и вычислительным системам все еще недостаточно внимания уделяется важности разработки требований (которая подразумевает как

собственно разработку, так и управление требованиями). Во-вторых, многие из тех, кто работает в области ПО, слишком увлечены техническими и процессными решениями наших задач. Мы иногда забываем, что выявление требований — и бо́льшая часть работы в проектах разработки ПО и систем вообще — основаны в первую очередь на взаимодействии людей. Не появилось никаких новых магических приемов для автоматизации этой деятельности, хотя на рынке имеются инструменты, позволяющие эффективно взаимодействовать географически распределенным людям.

Мы полагаем, что представленные во втором издании приемы разработки и управления требованиями по-прежнему работают и применимы к обширному множеству проектов по разработке ПО. Изобретательный бизнес-аналитик, менеджер или владелец продукта будет разумно адаптировать и масштабировать эти приемы в соответствии с особенностями конкретной ситуации. Новым в этом, третьем издании является глава о работе с требованиями в проектах гибкой разработки и разделы во многих других главах, описывающие, как применять и адаптировать описываемые в этих главах приемы при гибкой разработке.

Разработка ПО включает по крайней мере столько же общения, сколько и обычная работа с компьютером, но зачастую мы делаем акцент на работе с компьютером и не уделяем достаточно внимания общению. В этой книге описаны десятки способов, позволяющих реализовать это общение и помогающих разработчикам ПО, менеджерам, маркетологам и клиентам использовать на практике эффективные методы разработки требований к ПО. В книге описаны основные «отличные способы» формулирования требований, а не экзотичные или тщательно разработанные методологии, обещающие решить все проблемы, возникающие при формулировании требований. В многочисленных врезках я привожу реальные примеры, иллюстрирующие типичные ситуации, возникающие при формулировании требований.

С тех пор как в 1999 году вышло первое издание этой книги, каждый из авторов поработал над многочисленными проектами и провел сотни семинаров, посвященных требованиям к ПО для сотрудников коммерческих и правительственных организаций всех толков. Мы поняли, что эти способы годятся практически для любого проекта: маленьких и крупномасштабных, предусматривающих разработку новых программ и расширение уже имеющихся, силами локальных и распределенных команд и с использованием традиционных и гибких методов разработки. Кроме того, эти приемы не ограничиваются только областью разработки ПО, и вполне годятся для проектирования оборудования и систем. Как и в случае с любыми другими способами разработки ПО, чтобы понять, какие из способов формулирования требований подходят вам более всего, следует руководствоваться здравым смыслом и опираться на собственный опыт. Используйте описанные здесь приемы как инструменты, помогающие гарантировать, что вы эффективно общаетесь с нужными людьми в своих проектах.

Что дает эта книга

Из всех возможных способов совершенствования процесса разработки ПО наибольшее преимущество за формулированием требований. Мы описываем проверенные на практике способы, которые помогут вам:

- повысить качество требований к проекту на ранней стадии цикла разработки, что позволит снизить число доработок и повысить производительность;
- предоставлять высококачественные информационные системы и коммерческие продукты, которые решают поставленные задачи;
- управлять расползанием границ проекта и изменениями требований, обеспечивая контролируемость и отсутствие отклонений от цели;
- повысить удовлетворенность клиентов;
- снизить затраты на обновление, обслуживание и поддержку.

Наша цель — помочь вам усовершенствовать процессы выявления и анализа требований, написания и оценки спецификаций требований и управления требованиями на протяжении всего цикла разработки продукта. Описываемые нами приемы прагматичны и реалистичны. Мы использовали их неоднократно и всегда получали хорошие результаты.

Кому адресована эта книга

Она для тех, кто так или иначе вовлечен в сбор и формулирование требований к программному продукту. Основная аудитория — бизнес-аналитики и разработчики в проекте разработки независимо от того, работают они полный день или привлекаются к проекту время от времени. Более широкий круг читателей — архитекторы, дизайнеры, программисты, тестировщики ПО и другие члены команды, задача которых понять и удовлетворить чаяния клиентов, а также маркетологи и менеджеры по продуктам, которые должны проникнуться «духом» и особенностями продукта, чтобы сделать его в полной мере конкурентоспособным. Менеджеры проектов, отвечающие за своевременный выпуск, также найдут здесь для себя немало интересного: они узнают, как управлять показателями, связанными с требованиями к проекту, а также реагировать на изменение этих требований. Еще один сегмент аудитории — заинтересованные лица, участвующие в определении продукта, который соответствует их функциональным и бизнес-нуждам, а также потребностям в области качества. Настоящая книга поможет конечным пользователям, клиентам, создающим или заказывающим разработку ПО, а также многим другим заинтересованными лицам понять важность процесса формулирования требований и свое место в этом процессе.

Структура книги

Книга состоит из пяти частей. Первая часть «Требования к ПО: что, почему и кто» начинается с ряда определений. Если вы занимаетесь техническими вопросами, надеюсь, вы поделитесь концепциями главы 2, посвященной взаимодействию клиентов и разработчиков, со своими ключевыми клиентами. В главе 3 описано несколько десятков рекомендованных приемов создания и управления требованиями, а также процесс формулирования требований в целом. Глава 4 посвящена роли бизнес-аналитика.

Вторая часть «Разработка требований» начинается с описания способов для определения бизнес-требований в проекте. Другие главы этой части посвящены тому, как правильно выбрать представителей клиента, узнать у них требования и задокументировать пользовательские требования, бизнес-правила, функциональные требования, требования к данным и нефункциональные требования. В главе 12 обсуждается несколько моделей анализа, представляющих требования с разных точек зрения и дополняющих текстовую информацию, в главе 15 — применение прототипов ПО, позволяющих снизить риск выпуска продукта ненадлежащего качества. Прочие главы посвящены расстановке приоритетов, утверждению и оценке требований. Завершается вторая часть описанием, как требования влияют на другие аспекты работы над проектом.

Третья часть содержит совершенно новый материал, содержащий рекомендации по работе с требованиями в различных классах проектов: проектов гибкой разработки, проектов по доработке или замене систем, проектов, в которых используются готовые серийные пакетные решения, проектов, выполняемых сторонними организациями, проектов автоматизации бизнес-процессов, а также проектов встроенных и других систем реального времени.

В четвертой части «Управление требованиями» речь пойдет о принципах и способах управления требованиями, особое внимание уделяется технологиям, позволяющим реагировать на изменение требований. В главе 29 рассказано, как отслеживать изменение требований с самого начала до их реализации в продукте. Четвертая часть завершается описанием серийных утилит, расширяющих возможности разработки и управления требованиями к проекту.

Заключительная часть книги «Реализация процесса построения требований» поможет вам перейти от теории к практике. В главе 31 рассказано, как включить новые способы формулирования требований в имеющийся процесс разработки. Глава 32 посвящена рискам, связанным с требованиями к проекту. Приложение А позволит вам оценить применяемые способы формулирования требований и определить области, требующие совершенствования. В прочих приложениях представлены руководство по устранению проблем, возникающих при формировании требований, а также примеры задокументированных требований к проектам, чтобы вы могли увидеть, как это все работает на деле.

Примеры из практики

Методы, описанные в книге, мы иллюстрируем примерами, за основу которых взяты реальные проекты, это касается и информационной системы среднего размера под названием Chemical Tracking System. Не волнуйтесь — чтобы разобраться в этом проекте, знание химии вам не потребуется. Диалоги участников проектов из примеров разбавляют текст книги. Думаю, вне зависимости от того, что за ПО разрабатывает ваша команда, вы найдете эти диалоги интересными и полезными.

От принципов — к практике

Трудно представить, сколько энергии потребуется на преодоление препятствий, мешающих изменению и практическому применению новых знаний. Чтобы облегчить вам применение новых знаний на практике, каждую главу я заканчиваю разделом «Что дальше?», где перечислены конкретные действия, которые вы можете выполнить в отношении реального проекта. В разных главах есть шаблоны документов для определения требования, контрольные списки проверки, таблица приоритетов требований, описание процесса «изменение-контроль», а также полезные вещи для многих других процессов. Все это можно загрузить с веб-сайта этой книги по адресу: *http://aka.ms/SoftwareReq3E/files*.

Используйте эти материалы, чтобы применять полученные значения на практике. Начните с небольших усовершенствований, но именно сегодня, не откладывая на завтра.

Отдельные участники проекта с большой неохотой возьмутся за применение новых приемов работы с требованиями. Используйте материал книги для обучения коллег, клиентов и менеджеров. Напоминайте им о связанных с требованиями проблемах, возникавших при работе над предыдущими проектами, и обсуждайте потенциальные преимущества использования новых способов.

Не обязательно запускать новый проект, чтобы начать применять усовершенствованные способы формулирования требований. В главе 21 рассказывается, как использовать описываемые в книге способы в проектах по улучшению или замене существующей системы. Постепенное внедрение новых способов — сопряженный с незначительным риском подход к совершенствованию процесса разработки, который подготовит вас к использованию новых способов при работе над следующим крупным проектом.

Цель разработки требований — накопить требования, которые позволят проектировать приложения с *приемлемым* уровнем риска. Потратив на это достаточно времени, вы можете быть уверены, что свели к минимуму риск переделки продукта, создания негодного ПО или срыва сроков сдачи проекта. Эта книга научит вас, как объединить усилия нужных людей для разработки качественных требований к нужному продукту.

Часть I

Требования к ПО: что, почему и кто

Глава 1

Основы разработки требований к ПО

«Привет, Фил, это Мария из отдела персонала. У нас проблема с системой управления персоналом, которую вы разрабатывали для нас. Одна из сотрудниц изменила имя на Спаркл Старлайт, а система не принимает это изменение. Ты можешь помочь?»

«Она вышла замуж за парня по имени Старлайт?»

«Нет, она просто взяла другое имя, — ответила Мария. — В этом-то и проблема. Похоже, что мы можем менять имя только при изменении семейного положения».

«Да, я не предусмотрел такой возможности. Я не помню, чтобы и ты говорила мне о ней, когда мы обсуждали систему», — сказал Фил.

«Я полагала, ты знаешь, что люди могут законным образом менять имена, когда захотят, — ответила Мария. — Мы должны исправить это к пятнице, а то Спаркл не сможет обналичить чек. Ты сможешь исправить этот дефект к этому сроку?»

«Это не дефект, — резко парировал Фил. — Я и не знал, что вам нужна эта функциональность. Сейчас я занимаюсь новой системой оценки производительности системы, и скорее всего смогу исправить это в конце месяца, но не к пятнице. Приношу свои извенения. Но в следующий раз с такими просьбами обращайся пораньше и, пожалуйста, излагай их письменно».

«А что же я скажу Спаркл? — возмутилась Мария. — Она сильно расстроится, если не сможет обналичить чек».

«Эй, Мария, это не моя вина, — запротестовал Фил. — Если бы ты сказала мне с самого начала, что вам понадобится изменять имена в любое время, этого не произошло бы. Я не медиум и читать твои мысли не умею».

Сердитая и смирившаяся, Мария отрезала: «Это как раз то, из-за чего я ненавижу компьютерные системы. Позвони мне, как только сможешь исправить недостаток».

Если вы когда-либо бывали в шкуре клиента на переговорах, подобных этому, то знаете, как расстраиваются пользователи продукта, не позволяющего решать элементарные задачи. Не очень приятно находится в милости разработчиков, которые *может быть* удовлетворят ваш запрос. Разработчики тоже не очень довольны, когда узнают о функциональности, которую пользователи ожидали получить, только после развертывания системы.

Масса проблем с ПО возникает из-за несовершенства способов, которые люди применяют для сбора, документирования, согласования и модификации требований к ПО. Как в случае с Филом и Марией, проблемы могут возникать из-за неформального сбора информации, предполагаемой функциональности, ошибочных или несогласованных предположений, недостаточно определенных требований и бессистемного изменения процесса. Многие исследования показывают, что на ошибки, внесенные на этапе сбора требований, приходится от 40 до 50% всех дефектов, обнаруженных в программном продукте (Davis, 2005). Некорректные сведения от пользователей и недостатки определения и управления требованиями клиентов — основные причины провалов проектов. Но невзирая на эти сведения многие организации все еще применяют неэффективные методы сбора требований.

Нигде более, как на стадии сбора требований так тесно не связаны интересы всех заинтересованных в проекте лиц с успехом проекта. (Подробнее о заинтересованных лицах см. главу 2.) К заинтересованным в проекте лицам относятся клиенты, пользователи, бизнес-аналитики, разработчики и многие другие. Хорошо справившись с этой стадией процесса, вы получите отличный продукт, восхищенных заказчиков и удовлетворенных разработчиков. В противном случае вам грозит непонимание, разочарование и разногласия, которые подрывают веру в продукт и его ценность. Так как требования представляют собой основу для действий и разработчиков и менеджеров проекта, все заинтересованные в проекте лица должны быть вовлечены в создание этого документа.

Но разработка требований и управление ими — трудный процесс! Здесь нет быстрых или волшебных решений. Однако многие организации борются с одними и теми же трудностями, для преодоления которых мы можем найти общие приемы, годные в различных ситуациях. В этой книге описаны десятки таких приемов. Их рекламируют как средства построения абсолютно новых систем, однако они отлично подходят для улучшения, замены и реинжиниринга проектов (см. главу 21) и выбора коммерческих готовых решений (см. главу 22). Командам, которые выбирают итеративный процесс, следуя методике гибкой разработки (agile), необходимо знать, что же делать на каждом этапе (см. главу 20).

Из этой главы вы узнаете, как:

- разобраться с ключевыми терминами, применяемыми при сборе требований к ПО;
- различать требования к *продукту* и к *проекту*;
- различать требования по *разработке* и *управлению*;
- обнаруживать тревожные симптомы некоторых связанных с требованиями проблем, которые могут иногда возникать.

> **Внимание!** Мы используем термины «система», «продукт», «приложение» и «решение» для определения любого вида ПО или содержащего ПО компонента, который создается для внутрикорпоративного использования, на продажу или по заказу.

Проверка текущего состояния дел с требованиями

Чтобы проверить текущую ситуацию с требованиями в вашей организации, посмотрите сколько из следующих условий выполняются в вашем последнем проекте. Если в вашей ситуации применимы три или четыре элемента, эта книга для вас:

- Бизнес-цели, концепция и границы вашего проекта никогда четко не определялись.
- У клиентов никогда не хватало времени на работу над требованиями с аналитиками или разработчиками.
- Ваша команда не может напрямую взаимодействовать с непосредственными пользователями, чтобы разобраться с их потребностями.
- Клиенты считают все требования критически важными, поэтому они не разбивают их по приоритетам.
- В процессе работы над кодом разработчики сталкиваются с многозначностью и отсутствием информации, поэтому им приходится догадываться о многих вещах.
- Общение между разработчиками и заинтересованными лицами концентрируется на окнах и функциях интерфейса, а не на задачах, которые пользователи будут решать с использованием программного продукта.
- Ваши клиенты никогда не одобряли требования.
- Ваши клиенты одобрили требования к выпуску или итерации, после чего продолжили вносить в них изменения.
- Область действия проекта увеличилась вместе с принятием изменений в требования, но сроки был нарушены из-за отсутствия дополнительных ресурсов и отказа от удаления части функциональности.
- Затребованные изменения требований были утеряны, и никто не знает состояние запроса на указанные изменения.
- Клиенты запросили определенную функциональность и разработчики реализовали ее, но никто ее не использует.
- В конце проекта спецификация была выполнена, но бизнес-задачи не были выполнены, и клиент не удовлетворен.

Определение требований к ПО

Когда группа людей начинает обсуждать требования, они обычно начинают с проблемы терминологии. Разные эксперты, говоря об одном и том же документе, могут называть его пользовательскими требованиями, требованиями к ПО, бизнес-требованиями, функциональными требованиями, системными требованиями, требованиями к продукту или проекту, пользовательской точкой зрения, функцией или ограничением. Имена, которые они применяют к различным требованиям, также различаются. Заказчики зачастую считают, что определенные ими требования — это высокоуровневая концепция про-

дукта, предназначенная для разработчиков. Те, в свою очередь, полагают, что в отношении клиентов это детализированная разработка интерфейса пользователя. Такое многообразие ведет к сумятице и раздражающим проблемам коммуникации.

Особенности интерпретации требований

Десятки лет спустя после появления программирования для компьютеров специалисты по ПО все еще увлеченно спорят о том, что из себя представляет *требование*. Мы не станем участвовать в этих дебатах, а просто представим ряд определений, которые мы считаем полезными.

Консультант Брайан Лоренс предположил, что *требования* — это «нечто такое, что определяет выбор дизайна» (Lawrence, 1997). Это неплохое неформальное определение, потому что в эту категорию можно отнести много видов информации. И, в конце концов, весь смысл разработки требований заключается в принятии соответствующих проектировочных решений, которые в конечном итоге должны удовлетворить клиента. Другое определение требования заключается в том, что продукт должен обеспечивать выгоду заинтересованному лицу. Тоже неплохо, но не очень точно. Наше любимое определение сформулировано Иеном Соммервиллем и Питом Сойером (Ian Sommerville и Pete Sawyer, 1997):

> *Требования это спецификация того, что должно быть реализовано. В них описано поведение системы, свойства системы или ее атрибуты. Они могут служить ограничениями в процессе разработки системы.*

Это определение учитывает самые разные типы информации, которые в совокупности называются *требованиями*. Требования охватывают как видение пользователя, так и внешнее поведение системы, а также представление разработчика о некоторых внутренних характеристиках. Они включают как поведение системы в определенных условиях, так и свойства, которые делают систему полезной и даже доставляющей удовольствие конечным операторам.

> **Внимание!** Не рассчитывайте, что все заинтересованные лица вашего проекта разделяют общее представление о том, что же такое требования. Задайте определения с самого начала, чтобы вы все говорили на одном языке.

Определение термина «требование» в словаре

Специалисты по ПО используют термин «требование» не совсем в том же смысле, что и его значение в словаре, то есть как что-то требуемое и обязательное, потребность или необходимость. Люди иногда задаются вопросом, стоил ли вообще приоритизировать требования, ведь низкоприоритетные требования никогда не реализуются. Если это не очень нужная вещь, то ее и требованием-то назвать сложно. Возможно, но как тогда называть эту информацию? Если перенести требование из нынешнего проекта в неопределенный будущий выпуск, то можно ли его все равно считать требованием? Конечно да.

Требования к ПО включают измерение времени. Они могут относиться к настоящему времени — в этом случае они описывают текущие функции системы. Или могут относиться к ближайшему (высокоприоритетные), среднему (средреприоритетные) или гипотетическому (низкоприоритетные) будущему. Они могут даже относиться к прошлому времени, когда описывают запросы, которые были ранее определены, а затем отброшены. Не стоит тратить время на споры является ли что-то требованием или нет, даже если вы знаете, что оно скорее всего не будет реализовано по какой-то серьезной причине. Это требование и точка.

Уровни и типы требований

Из-за такого большого числа разных типов информации требований нам требуется единый набор описаний, изменяющих слишком перегруженный термин *требование*. В этом разделе приводятся определения, которые будут использоваться для терминов, наиболее часто применяемых у такой сфере, как разработка требований (см. табл. 1-1).

Табл. 1-1. Информация о некоторых типах требований

Понятие	Определение
Бизнес-требование	Высокоуровневая бизнес-цель организации или заказчиков системы
Бизнес-правило	Политика, предписание, стандарт или правило, определяющее или ограничивающее некоторые стороны бизнес-процессов. По своей сути это не требование к ПО, но оно служит источником нескольких типов требований к ПО
Ограничение	Ограничение на выбор вариантов, доступных разработчику при проектировании и разработке продукта
Внешнее требование к интерфейсу	Описание взаимодействия между ПО и пользователем, другой программной системой или устройством
Характеристика	Одна или несколько логически связанных возможностей системы, которые представляют ценность для пользователя и описаны рядом функциональных требований
Функциональное требование	Описание требуемого поведения системы в определенных условиях
Нефункциональное требование	Описание свойства или особенности, которым должна обладать система, или ограничение, которое должна соблюдать система
Атрибут качества	Вид нефункционального требования, описывающего характеристику сервиса или производительности продукта
Системное требование	Требование верхнего уровня к продукту, состоящему из многих подсистем, которые могут представлять собой ПО или совокупность ПО и оборудования
Пользовательское требование	Задача, которую определенные классы пользователей должны иметь возможность выполнять в системе, или требуемый атрибут продукта

Требования к ПО состоят из трех уровней — бизнес-требования, пользовательские и функциональные требования. Вдобавок в каждой системе есть свои нефункциональные требования. Модель на рис. 1-1 иллюстрирует способ представления этих типов требований. Как гласит знаменитое высказывание статистика Джорджа Е. П. Бокса (George E. P. Box), «В конечном счете все модели врут, но некоторые оказываются полезными» (Box и Draper, 1987). Это, конечно же, применимо к рис. 1-1. Как и все модели, она не полная, но схематично показывает организацию требований.

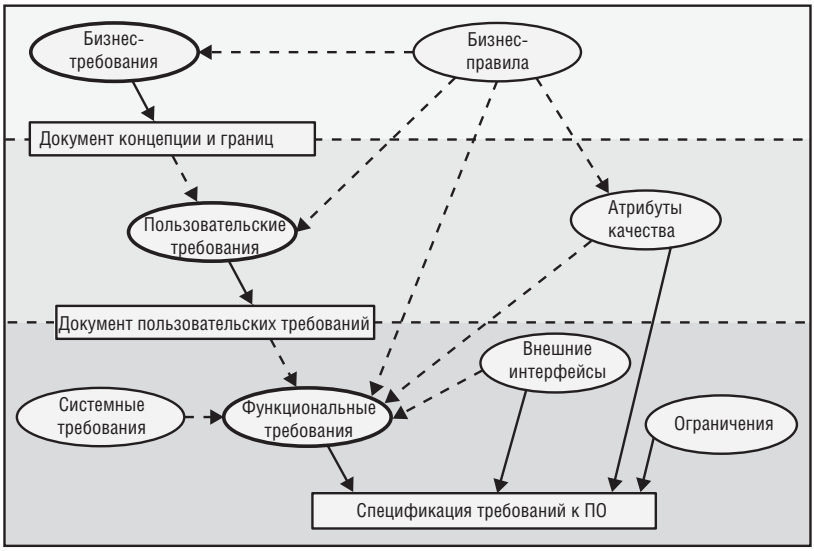


Рис. 1-1. Взаимосвязи нескольких типов информации для требований. Сплошные линии означают «содержатся в», а пунктирные — «являются отправной точкой» или «влияют на»

> **Внимание!** Хотя мы в этой книге и называем требования «документами», как на рис. 1-1, это не обязательно традиционные бумажные или электронные документы. Их можно считать просто контейнерами, в которых хранится знание о требованиях. Такой контейнер может быть традиционным документом или же электронной таблицей, набором диаграмм, базой данных, средством управления требованиями или сочетанием всех этих артефактов. Для удобства мы будем называть документом любой такой контейнер. Мы предоставим шаблоны, которые определяют типы информации, которые стоит хранить в каждой из таких группировок независимо от того, в какой форме они хранятся. Как называть такие артефакты не так важно, как добиться согласия в компании относительно их имен, какие виды информации в них содержатся, а также как эта информация организована.

Овалы обозначают типы информации требований, а прямоугольники — документы, в которых хранится эта информация. Сплошные линии указывают, что в указанном документе хранится информация определенного типа. (Бизнес-правила и системные требования хранятся отдельно от требований к ПО, обычно соответственно в каталоге бизнес-правил или в спецификации системных требований.) Пунктирная линия указывает, что информа-

ция одного типа является источником или влияет на информацию другого типа или на требование. На этой схеме не показаны требования к данным. Функции манипулируют данными, поэтому требования к данным могут присутствовать на всех трех уровнях. В главе 7 приводится много примеров различных типов информации требований.

Бизнес-требования (business requirements) описывают, *почему* организации нужна такая система, то есть цели, которые организация намерена достичь с ее помощью. Основное их содержание — бизнес-цели организации или клиента, заказывающих систему. Допустим, что авиакомпания хочет на 25% снизить затраты на сотрудников у стойки в аэропорту. В процессе достижения этой цели может возникнуть идея поставить терминал, который пассажиры будут использовать для самостоятельной регистрации в аэропорту. Как правило, бизнес-требования высказывают те, кто финансируют проект, покупатели системы, управляющий реальными пользователями, отдел маркетинга или ответственный за концепцию продукта. Мне нравится записывать бизнес-требования в форме документа *о концепции и границах* (vision and scope document). К другим руководящим документам, которые еще иногда используют в этом качестве, относят устав проекта (project charter), вариант использования (*business case*) или документ рыночных требований (market requirements document). Определение бизнес-требований описано в главе 5. Для целей этой книги мы предполагаем, что бизнес-потребность (business need) или рыночная возможность (market opportunity) уже определена.

Пользовательские требования (user requirements) описывают цели или задачи, которые пользователи должны иметь возможность выполнять с помощью продукта, который в свою очередь должен приносить пользу кому-то. Область пользовательских требований также включает описания атрибутов или характеристик продукта, которые важны для удовлетворения пользователей. К отличным способам представления этого вида требований относятся варианты использования (Kulak и Guiney, 2004), пользовательские истории (Cohn, 2004) и таблицы «событие — отклик». В идеале эту информацию предоставляют реальные представители пользователей. Пользовательские требования описывают, *что* пользователь должен иметь возможность делать с системой. Примером сценария использования является регистрация на рейс с использованием веб-сайта или терминала в аэропорту. Будучи сформулированным в виде пользовательской истории, то же пользовательское требование может звучать так: «Как пассажир я хочу зарегистрироваться на рейс, чтобы можно было сесть на самолет». Важно помнить, что в большинстве проектов есть несколько классов пользователей, а также других участников, потребности которых тоже надо выявить. Этот уровень модели описывается в главе 8. Некоторые люди используют более широкий термин «требования заинтересованных лиц» для обозначения того факта, что требования будут предоставляться различными заинтересованными лицами, отличными от прямых пользователей системы. Это конечно же верно, но на данном уровне нам нужно понять, что реальные пользователи хотят достичь с помощью продукта.

Функциональные требования (functional requirements) определяют, каким должно быть поведение продукта в тех или иных условиях. Они определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи (пользовательские требования) в рамках бизнес-требований. Такое соотношение между тремя уровнями требований жизненно важно для успеха проекта. Функциональные требования описываются в форме традиционных утверждений со словами «*должен*» или «*должна*»: «У пассажира должна быть возможность распечатать посадочные талоны на все рейсы, на которые он зарегистрировался» или «Если в профиле пассажира не указаны предпочтения по выбору места, система резервирования должна сама назначить ему место».

Бизнес-аналитик[1] документирует функциональные требования в *спецификации требований к ПО* (software requirements specification, SRS), где описывается так полно, как необходимо, ожидаемое поведение системы. Спецификация требований к ПО используется при разработке, тестировании, гарантии качества продукта, управлении проектом и в связанных с проектом функциях. Этот артефакт называют по-разному: документ бизнес-требований, функциональная спецификация, документ требований и т. п. Спецификация требований к ПО может представлять собой отчет, сгенерированный на основе информации, хранимой в средстве управления требованиями. Так как этот термин принят в отрасли, мы будем в этой книге использовать обозначение «SRS» (ISO/IEC/IEEE 2011). Подробнее об SRS см. главу 10.

Термин *системные требования* (system requirements) описывает требования к продукту, который содержит многие компоненты или подсистемы (ISO/IEC/IEEE 2011). В этом смысле «система» это не просто любая информационная система. Говоря о системе, мы подразумеваем программное обеспечение или подсистемы ПО и оборудования. Люди и процессы тоже часть системы, поэтому определенные системные функции могут распространяться и на людей. Некоторые используют термин «системные требования» для обозначения подробных требований к программной системе, но в этой книге мы не используем этот термин в таком смысле.

Хороший пример «системы» — рабочее место кассира в супермаркете. В нем есть сканер штрих-кода, интегрированный с весами, а также ручной сканер штрих-кода. У кассира есть клавиатура, монитор и выдвижной ящик-касса. Есть также устройство чтения кредитных карточек и клавиатура для ввода ПИН-кода и чтения карт постоянного покупателя. На рабочем месте может быть до трех принтеров: для печати чека, квитанции кредитной карты и купонов на скидку. Все эти устройства взаимодействуют под управлением программного обеспечения. На основе требований к системе или продукту в целом бизнес-аналитик формулирует конкретную функциональность, кото-

[1] «Бизнес-аналитик» — это роль в проекте, которая прежде всего отвечает за действия по работе с требованиями в проекте. У роли бизнес-аналитика есть и другие имена. Подробнее о роли бизнес-аналитика см. главу 4.

рую должны поддерживать тот или иной компонент или подсистема, а также интерфейсы взаимодействия между ними.

Бизнес-правила (business rules) включают корпоративные политики, правительственные постановления, отраслевые стандарты и вычислительные алгоритмы. Как вы увидите в главе 9, бизнес-правила не являются сами требованиями к ПО, потому что они находятся за пределами любой системы ПО. Однако они часто налагают ограничения, определяя, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда, как в случае с корпоративными политиками безопасности, бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности. Следовательно, вы можете отследить происхождение конкретных функциональных требований вплоть до соответствующих им бизнес-правил.

В дополнение к функциональным требованиям спецификация SRS содержит нефункциональные. *Атрибуты качества* (quality attributes) еще называют параметрам качества, требованиями по уровню обслуживания и т. п. Они представляют собой описание различных измерений характеристик продукта, которые важны для пользователей или для разработчиков и тех, кто будет обслуживать систему, таких как производительность, доступность и переносимость. Другие нефункциональные требования описывают *внешние интерфейсы* между системой и внешним миром. Речь идет о подключениях к другим программным системам, аппаратным устройствам и пользователям, а также коммуникационные интерфейсы. *Ограничения* (constraints) проектирования и реализации накладывают границы на возможности выбора разработчика при проектировании продукта.

Если они нефункциональные, то что они из себя представляют?

Многие годы требования к программным продуктам в целом классифицировались как функциональные и нефункциональные. С функциональными требованиями все ясно: они описывают наблюдаемое поведение системы в различных обстоятельствах. Но многим не нравится термин «нефункциональные». Это прилагательное говорит, чем *не являются* требования, но не говорит, чем они *являются*. Мы понимаем проблему, но не можем предложить идеального решения.

Требования, отличные от функциональных, могут описывать не *что* система делает, а *как хорошо* она это делает. Они могут описывать важные характеристики или свойства системы. К ним относятся доступность, легкость и простота использования, производительность и другие характеристики системы, как описано подробнее в главе 14. Некоторые люди считают нефункциональные требования тем же, что атрибуты качества, но это слишком узкое понимание. Например, ограничения проекта или реализации также являются нефункциональными требованиями, как требования внешних интерфейсов.

Другие нефункциональные требования описывают среду, в которой работает система, например платформу, переносимость, совместимость и ограничения. Многие продукты также должны подчиняться определенным правилам, требованиям регулирующих органов или требовать сертификации. Такими могут быть требования по локализации для продуктов, в которых должны учитываться региональные стандарты, языки, законы, валюты, терминология, орфография и другие характеристики пользователей. Хотя такие требования определяются с использованием нефункциональной терминологии, бизнес-аналитик на их основе может определить много функций, чтобы система обладала всеми необходимыми свойствами и вела себя соответствующим образом в разных ситуациях.

Несмотря на описанные ограничения, в этой книге мы будем придерживаться термина «нефункциональные требования» за неимением подходящего всеобъемлющего альтернативного термина. Не надо волноваться о точности названия всей подобной информации — лучше позаботьтесь, чтобы она вошла в ваши действия по выявлению и анализу требований. Можно создать продукт, обладающий всей требуемой функциональностью, но пользователи могут невзлюбить его за то, что тот не соответствует их (обычно невысказанным) ожиданиям по качеству.

Характеристика (feature) — это набор логически связанных функциональных требований, которые представляют ценность для пользователя и удовлетворяют бизнес-цели. Желательные характеристики продукта, которые перечисляет клиент, не эквивалентны тем, что входят в список необходимых для решения задач пользователей. В качестве примеров характеристик продуктов можно привести избранные страницы или закладки веб-браузера, средства проверки орфографии, запись макрокоманды, автоматическое обновление определений вирусов в антивирусной программе. Характеристики могут охватывать множество пользовательских требований, и для каждого варианта необходимо, чтобы множество функциональных требований было реализовано для выполнения пользователем его задач. Рис. 1-2 иллюстрирует *дерево функций* (feature tree) — модель анализа, которая показывает, как функцию можно разложить на иерархию более мелких функций, которые связаны с конкретными пользовательскими требованиями и ведут к определению наборов функциональных требований (Beatty и Chen, 2012).

Чтобы вы лучше восприняли некоторые из различных видов требований, проанализируем проект по разработке следующей версии текстового редактора. Бизнес-требование может звучать так: «Увеличить продажи за пределами США на 25% за следующие полгода». В отделе маркетинга выяснили, что в продуктах-конкурентах есть только англоязычные средства проверки орфографии, поэтому было принято решение включить возможность проверки орфографии на других языках. Соответствующие требования пользователей могут содержать задачи вроде такой: «Выберите язык проверки орфографии», «Найдите орфографические ошибки» или «Добавьте слово в словарь».

Проверка грамматики имеет множество индивидуальных функциональных требований, которые имеют дело с такими операциями, как поиск и выделение слов с ошибками, автоисправление, отображение вариантов замены и замена слова с ошибкой корректным вариантом по всему тексту. Требования по *удобству использования* (usability) определяют, как нужно локализовать ПО для использования с различными языками и наборами символов.

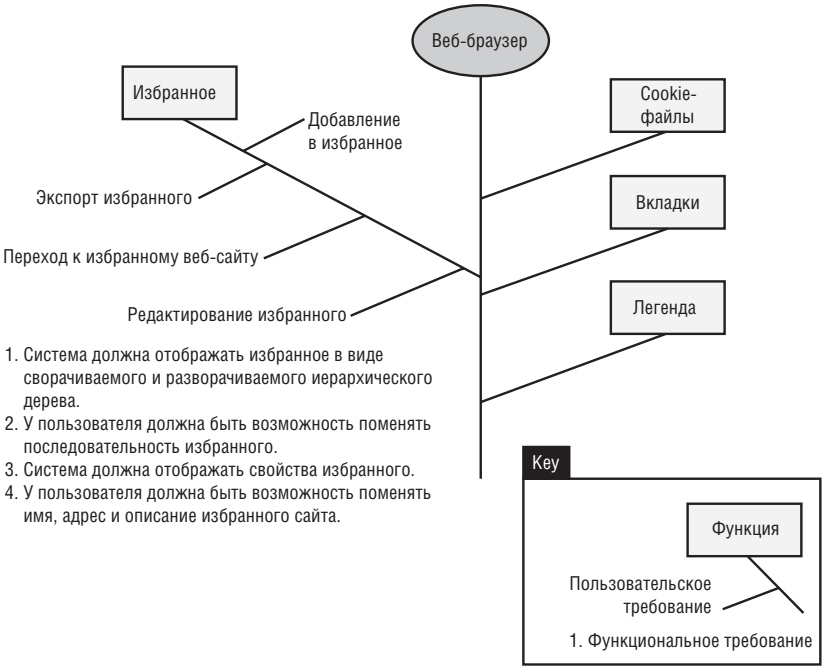


Рис. 1-2. Взаимоотношения между функциями и пользовательским и функциональными требованиями

Три уровня требований

На рис. 1-3 показано, как различные заинтересованные лица могут участвовать в выявлении трех уровней требований. В разных организациях используют разные имена для ролей, участвующих в этой деятельности; подумайте, кто выполняет эту работу в вашей организации. Имена ролей часто различаются в зависимости от того, является ли подразделение, разрабатывающее продукт, внутренним отделом организации или компанией, создающей ПО для коммерческого использования.

На основе выявленной бизнес-потребности, требования рынка или интересной концепции нового продукта менеджеры и сотрудники отдела маркетинга определяют бизнес-требовния для ПО, которые помогут компании работать эффективнее (для информационных систем) или успешно конкурировать на рынке (для коммерческих продуктов). В корпоративной среде

после этого аналитики обычно работают с представителями пользователей для определения пользовательских требований. В компаниях, разрабатывающих коммерческие продукты, часто назначают менеджера продукта, который определяет, какие функции должны включаться в новый продукт. Каждое пользовательское требование должно быть сопоставлено бизнес-требованию. На основе пользовательских требований аналитик или менеджер продукта определяет функции, которые дадут возможность пользователям выполнять их задачи. Разработчикам необходимы функциональные и нефункциональные требования, чтобы создавать решения с желаемой функциональностью, не выходя за рамки налагаемых ограничений. Тестировщики определяют, как проверять правильность реализации требований.

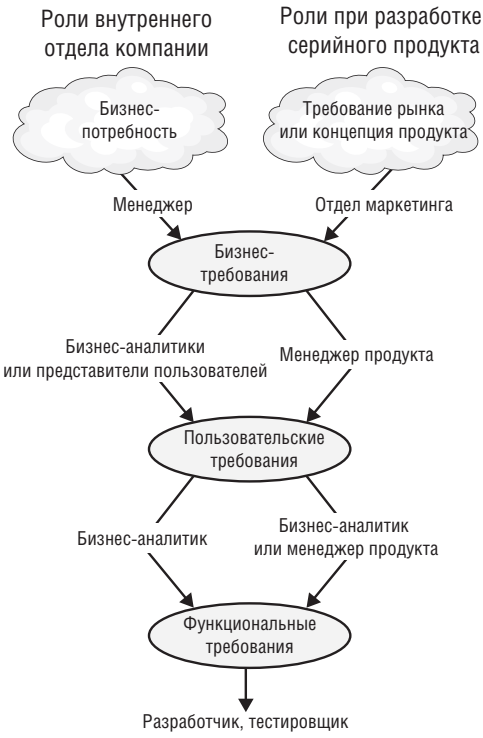


Рис. 1-3. Пример участия различных заинтересованных лиц в разработке требований

Важно понимать ценность письменной фиксации жизненно важных требований в доступной для совместного использования форме, а не сведений, передающихся от человека к человеку изустно. Однажды я работал над проектом, в котором часто менялись команды разработчиков. Основной заказчик был недоволен тем, что из каждой новой команды к нему приходили со словами: «Нам надо поговорить о требованиях». Его реакция на этот запрос звучала так: «Я уже рассказал вашим предшественникам о моих требованиях. Так что просто займитесь созданием системы!» К сожалению, никто не потрудился задокументировать никаких требований, поэтому каждой новой

команде приходилось начинать с нуля. По крайней мере безответственно заявлять, что у вас «есть требование», если на самом деле у вас несколько сообщений электронной почты и голосовой почты, ряд записок-наклеек, протоколы совещаний и туманные воспоминания о разговорах с заказчиком. Аналитик должен выработать осмотрительный подход для определения, насколько полной должна быть документация требований в том или ином проекте.

На рис. 1-1 были показаны три главных документа требований: документ концепции и границ, документ пользовательских требований и спецификация программных требований. Не всегда обязательно создавать эти три отдельных документа в каждом проекте. Часто имеет смысл объединять часть информации, особенно в небольших проектах. Однако надо понимать, что эти три документа содержат разную информацию, разрабатываются на разных этапах проекта, возможно даже разными людьми с разными целями и разной целевой аудиторией.

Модель на рис. 1-1 показывает простой «сверху вниз» поток информации требований. В реальности возможно наличие циклов и итераций пользовательских, функциональных и бизнес-требований. Каждый раз когда кто-то предлагает ввести новую функцию, пользовательское требование или улучшение функциональности, аналитик должен задаться вопросом: «Укладывается ли это в рамки проекта?» Если ответ положительный, требование должно присутствовать в спецификации. В противном случае требования в спецификации быть не должно, по крайней мере в текущем выпуске или итерации. Третий возможный ответ звучит так: «Нет, но это поддерживает бизнес-цели, поэтому должно быть в спецификации». В этом случае, ответственный за область действия проекта — куратор, менеджер или ответственный за проект, должен решить, нужно ли расширять рамки текущего проекта или итерации, чтобы включить новое требование. Это бизнес-решение, которое оказывает влияние на график и бюджет проекта и может требовать пожертвовать другими возможностями. Эффективный процесс управления изменениями, включающий анализ последствий, гарантирует, что «правильные люди» принимают информированные бизнес-решения о том, какие изменения следует принять, а также что учитываются сопутствующие затраты времени или ресурсов или компромиссы в выборе функциональности.

Требования к продукту и требования к проекту

До этого момента мы обсуждали требования, описывающие свойства программной системы, которую планируется построить. Назовем их требованиями к *продукту*. Для проекта, как правило, создаются требования других типов: документ, где описана среда разработки, ограничения бюджета, руководство пользователя или требования для выпуска продукта и продвижения его в поддерживаемую среду. Это требования к *проекту*, но не к *продукту*. Спецификация требований содержит требования к продукту и не содержит

деталей дизайна или реализации (кроме известных ограничений), данных о планировании проекта, сведений о тестировании и тому подобной информации. Удалите указанные элементы из требований, чтобы из этого документа было абсолютно ясно, что надлежит построить команде разработчиков. К требованиям проекта относятся:

- физические ресурсы, необходимые команде разработки, такие как рабочие станции, специальные аппаратные устройства, тестовые лаборатории, средства и оборудование тестирования, командные комнаты и оборудование для видеоконференций;
- потребности в обучении персонала;
- пользовательская документация, включая обучающие материалы, пособия, справочные руководства и информация о выпусках ПО;
- документация для поддержки, такая как ресурсы службы технической поддержки, а также информация о техническом обеспечении и обслуживании аппаратных устройств;
- инфраструктурные изменения, которые необходимо внести в рабочую среду;
- требования и процедуры для выпуска продукта, установки в рабочей среде, конфигурирования и тестирования;
- требования и процедуры для перехода со старой на новую систему, например требования по переносу и преобразованию данных, по настройке безопасности, переносу производства и обучению для восполнения недостатка квалификации — это требования иногда называют *требованиями по переходу* (transition requirements) (IIBA 2009);
- требования по сертификации продукта и его соответствия требованиям регулирующих органов;
- скорректированные политики, процессы, организационные структуры и аналогичные документы;
- сорсинг, приобретение и лицензирование ПО сторонних производителей и компонентов оборудования;
- требования по бета-тестированию, производству, упаковке, маркетингу и дистрибуции;
- соглашения об уровне обслуживания с клиентами;
- требования по правовой защите (патенты, товарные знаки или авторское право) интеллектуальной собственности, связанной с разрабатываемым ПО.

Эти требования к проекту не будут рассмотрены в этой книге. Это не значит, что они не важны, — просто они находятся за рамками нашей тематики, посвященной разработке и управлению требованиям к программному продукту. Определение этих требований к проекту является совместной ответственностью бизнес-аналитика и менеджера проекта. Они часто возникают при сборе требований к продукту. Информацию требований к проекту лучше всего хранить в плане управления проектом, который должен описывать все

ожидаемые операции и результаты проекта.

В частности бизнес-приложения люди иногда называют «решением», которое охватывает как требования к продукту (за которые отвечает бизнес-аналитик), так и требования к проекту (за которые отвечает менеджер проекта). Они могут употреблять термин «рамки решения» в смысле «все, что нужно сделать для успешного завершения проекта». Однако в этой книге мы сосредоточимся на требованиях к продукту, что бы это ни было — коммерческий программный продукт, аппаратное устройство с встроенным ПО, корпоративная информационная система, ПО для государственных учреждений и т. п.

Разработка и управление требованиями

Путаница в терминологии возникает, как только речь заходит о требованиях, и затрагивает даже то, что именно называть этим словом. Некоторые авторы называют всю эту предметную область *разработкой технических условий* (нам такой подход ближе всего), вторые употребляют термин *управление требованиями* (requirements management), а третьи считают эту деятельность подмножеством более общей предметной области бизнес-анализа.

Мы считаем полезным разделить область разработки технических условий на *разработку требований* (requirements development) — подробнее об этом в части 2 этой книги — и *управление требованиями* (requirements management) — см. часть IV, как показано на рис. 1-4. Независимо от методологии проекта — водопадная, поэтапная, итеративная, гибкая или смешанная — есть вещи, которые надо выполнить в отношении всех требований. В зависимости от методологии эти операции могут выполняться в разное время жизненного цикла проекта и с разным уровнем глубины и детализации.

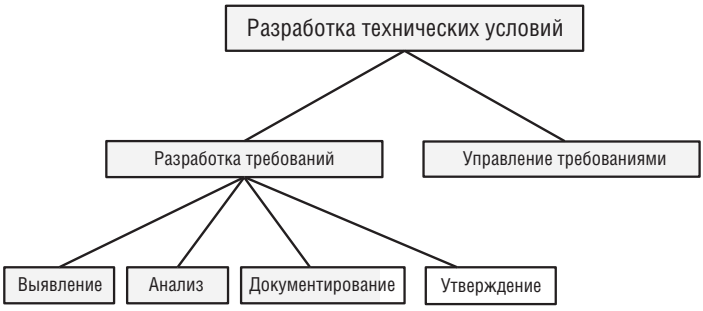


Рис. 1-4. Составные части предмета разработки технических условий

Разработка требований

Как показано на рис. 1-4, мы подразделяем разработку технических условий на *выявление* (elicitation), *анализ* (analysis), *документирование* (specification) и *утверждение* (validation) (Abran и другие, 2004). В эти составные части входят все действия, включающие сбор, оценку, документирование и утверждение требований для ПО. Далее описаны основные действия в каждой из

составных частей.

Выявление и сбор требований

Выявление и сбор требований (elicitation) охватывает все действия, связанные с выявлением требований, такие как интервью, совещания, анализ документов, создание прототипов и другие. К ключевым действиям относятся:

- Определение классов ожидаемых пользователей продукта и других заинтересованных лиц.
- Понимание задач и целей, а также бизнес-целей, которым соответствуют эти задачи.
- Изучение среды, в которой будет использоваться новый продукт.
- Работа с отдельными людьми, которые представляют каждый класс пользователей, чтобы понять их потребности и ожидания в отношении качества.

На что ориентироваться: на использование или на продукт?

При сборе требований обычно используется подход, ориентированный на использование, или подход, ориентированный на продукт, хотя возможны и другие стратегии. В ориентированном на использование подходе упор делается на понимание и исследование задач пользователей, и на основе этой информации выводится необходимая функциональность системы. Ориентированный на продукт подход сфокусирован на определение функций, которые, как ожидается, приведут к успеху на рынке или успеху бизнеса компании. В стратегиях, ориентированных на продукт, есть риск реализовать функции, которые не будут активно использоваться, несмотря на то, что во время сбора требований они казались очень нужными. Мы рекомендуем сначала изучить бизнес-цели и цели пользователей, а затем на основе этой информации определить нужные функции и характеристики продукта.

Анализ

Анализ требований (analyzing requirements) подразумевает получение более обширного и точного понимания всех требований и представление наборов требований в различном виде. Далее перечислены основные действия:

- анализ информации, полученной от пользователей, чтобы отделить их задачи от функциональных и нефункциональных требований, бизнес-правил, предполагаемых решений и другой информации;
- разложение высокоуровневых требований до нужного уровня детализации;
- выведение функциональных требований из информации других требований;
- понимание относительной важности атрибутов качества;
- распределение требований по компонентам ПО, определенным в систем-

ной архитектуре;
- согласование приоритетов реализации;
- выявление пробелов в требованиях или излишних требований, не соответствующих заданным рамкам.

Документирование

Документирование требований предусматривает представление и хранение совокупного знания о требованиях постоянным и хорошо организованным способом. К ключевому действию относится:
- преобразование собранных потребностей пользователей в письменные требования и диаграммы, пригодные для понимания, анализа и использования целевой аудиторией.

Утверждение

Утверждение требований (requirements validation) должна подтвердить правильность имеющегося набора требований, которые позволят разработчикам создать решение, удовлетворяющее бизнес-целям. Далее перечислены основные действия:
- проверка задокументированных требований для устранения всех недостатков до принятия требований группой разработки;
- разработка приемочных тестов и критериев, которые должны подтвердить, что созданный на основе требований продукт будет отвечать потребностям заказчика и удовлетворять поставленным бизнес-целям.

Итерация — ключевое условия успеха разработки. При планировании нужно предусмотреть не один цикл проверки требований для поступательного уточнения деталей высокоуровневых требований и подтверждения правильности пользователями. На это может уходить много времени, и подобная деятельность может быть не очень захватывающей, но это неизбежная процедура разрешения всех неясностей при определении новой программной системы.

> **Внимание!** Вы никогда не сможете создать идеальные требования. С практической точки зрения цель разработки требований — накопить общее понимание требований, достаточно хорошее для создания очередной порции продукта — будь то один или сто процентов, чтобы продолжить работу при разумном уровне риска. Основной риск заключается в наличии большого объема незапланированных недоделок, потому что команда недостаточно глубоко разобралась с требованиями к очередному этапу работы перед началом проектирования и разработки.

Управление требованиями

К действиям по управлению требованиями относятся:
- определение основной версии требований, моментальный снимок, который представляет согласованный, проверенный и одобренный набор функциональных и нефункциональных требований, обычно для конкрет-

ного выпуска продукта или итерации разработки;

- оценка влияния предлагаемых требований и внедрение одобренных изменений в проект управляемым образом;
- обновление планов проекта в соответствии с изменениями в требованиях;
- обсуждение новых обязательств, основанных на оцененном влиянии изменения требований;
- определение отношений и зависимостей, существующих между требованиями;
- отслеживание отдельных требований до их проектирования, исходного кода и тестов;
- отслеживание состояния требований и действий по изменению на протяжении всего проекта.

Предмет управления требованиями заключается не в предотвращении изменении или усложнении их внесения — задача состоит в предугадывании и приспосабливании к ожидаемым реальным изменениям, чтобы снизить их разрушительное влияние на проект.

На рис. 1-5 показан другой способ разделения областей разработки требований и управления ими. В этой книге описаны десятки специальных приемов выявления требований, их анализа, документации, проверки и управления.

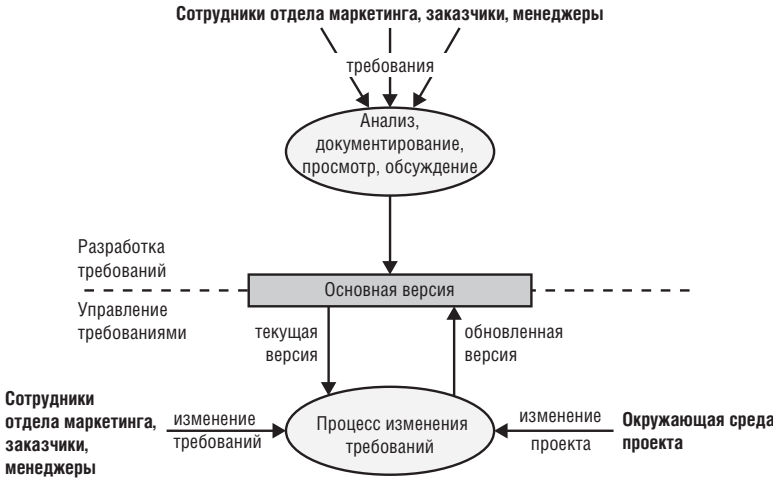


Рис. 1-5. Разделение областей разработки требований и управления ими

Каждый проект имеет требования

Фредерик Брукс выразительно определил критическую роль требований в проекте разработки ПО в классическом эссе 1987 года «No Silver Bullet: Essence and Accidents of Software Engineering»:

«Самая сложная часть построения систем ПО — решить точно, что же создавать. Никакая другая часть концептуальной работы не явля-

ется такой трудной, как выяснение деталей технических требований, в том числе и взаимодействие с людьми, механизмами и иными системами ПО. Никакая другая часть работы так не портит результат, если она выполнена плохо. Никакая другая часть не дает более трудные для исправления ошибки».

Каждая содержащая ПО система имеет пользователей, которые полагаются на нее. Время, которое тратится на выяснение потребностей пользователей, представляет собой высокоэффективную инвестицию в успех проекта. Если в команде проекта не регистрируют в письменном виде требования, с которыми заказчики уже согласились, как могут они удовлетворить этих заказчиков?

Зачастую невозможно — или не нужно — полностью определить требования до начала проектирования и реализации. В этом случае действуйте итеративно и постепенно: разрабатывайте одну порцию требований за раз и обязательно дожидайтесь ответной реакции заказчика, прежде чем приступать к следующему циклу. Это суть разработки по гибкой методологии — выяснение ровно такого объема требований, чтобы выполнить разумную приоритизацию и планирование выпуска, чтобы команда могла максимально быстро начинать создавать ценное ПО. Это не может служить оправданием написания кода до изучения требований перед следующим шагом. Итерации при кодировании стоят гораздо дороже, чем при разработке концепций.

Люди иногда предпочитают не тратить время на написание требований, но этот этап не самый сложный. Самое трудное — *выявить* эти требования. Первоначально написание требований представляет собой процесс выяснения, разработки и расшифровки данных. Четкое понимание требований к продукту дает вам уверенность, что ваша команда работает над теми проблемами, над которыми нужно, и создает лучшее их решение. Не зная, что собой представляют требования, вы не сможете определить момент окончания проекта, установить, достигнуты ли цели, или выбрать компромиссное решение, когда придется корректировать границы проекта. Вместо того, чтобы отказываться от затрат времени на создание требований, откажитесь от потерь денег из-за недостаточного внимания к требованиям в проекте.

Когда плохие требования появляются у хороших людей

Основное следствие проблем с требованиями — переделка того, что, как вы думаете, уже готово. На это расходуется от 30 до 50% общего бюджета разработки (Shull, et al., 2002; GAO, 2004), а ошибки в требованиях стоят от 70 до 85% стоимости переделки (Leffingwell, 1997). Небольшие переделки повышают ценность и улучшают продукт, но очень большой объем переделок не несет ничего кроме растраты ресурсов и разочарования. Вообразите, насколько изменилась бы ваша жизнь, если бы вы сократили объем переделок

наполовину! Члены вашей команды могли бы создавать ПО быстрее и даже приходить домой вовремя. Создание более качественных требований это инвестиции, а не затраты.

Гораздо дороже исправить дефекты, которые найдены позднее в проекте, чем сразу после создания. Допустим, что стоимость нахождения и устранения дефекта требования в процессе работы над ним составляет один доллар (по относительной шкале). Если же ошибка обнаружится во время проектирования, придется потратить доллар на исправление ошибки в требовании и еще 2-3 доллара на переделку проекта, основанного на неправильном требовании. Но представьте, что никто не обнаружил ошибку, пока не позвонил пользователь и не пожаловался на нее. Стоимость устранения дефекта зависит от типа системы и может достигать на нашей относительной шкале 100 или больше долларов (Boehm, 1981; Grady, 1999; Haskins, 2004). Один из клиентов, которому я оказывал консалтинговые услуги, определил, что в среднем трудозатраты на обнаружение и исправление дефекта в их информационной системе составляли 200 долларов, при этом использовалась специальная методика проверки ПО, один из видов дружественной проверки (Wiegers, 2002). С другой стороны, исправление ошибки, обнаруженной пользователем, обходилось в *4200 долларов* — в 21 раз дороже. Предотвращение ошибок в требованиях и обнаружение их на самых ранних этапах сильно снижает объем переделок.

Недостатки в требованиях представляют собой угрозу успеху проекта, где *успех* означает выпуск продукта, который удовлетворяет ожиданиям пользователей по качеству и функциональности при соблюдении бюджета и графика проекта. В главе 23 рассказано, как управлять такими рисками, чтобы они не привели к крушению проекта. Некоторые из наиболее общих рисков описаны далее в этой главе.

Недостаточное вовлечение пользователей

Заказчики зачастую не понимают, почему так важно тщательно собрать требования и обеспечить их качество. Разработчики не всегда придают значение вовлечению пользователей в процесс скорее всего из-за того, что они считают, что все уже знают о потребностях пользователей. В некоторых случаях трудно добраться до людей, которые непосредственно будут иметь дело с продуктом, а те, кто заменяет пользователей, не всегда понимают, что тем нужно в реальности. Недостаточное вовлечение пользователей ведет к обнаружению ошибок в требованиях на поздних стадиях проекта, а значит, к задержке завершения проекта.

Еще один риск, связанный с недостаточным вовлечением пользователей, особенно при анализе и проверке требований, заключается в том, что бизнес-аналитик может не понять и неправильно задокументировать реальные бизнес-требования или потребности клиента. Иногда бизнес-аналитик идет по пути определения того, что кажется «идеальными» требованиями, а разработчики реализуют их, но никто не использует решение, потому что

бизнес-задача была понята неправильно. Продолжающиеся совещания с пользователями могут помочь снизить риск, но если пользователи проверят требования недостаточно тщательно, у вас все равно могут быть проблемы.

Небрежное планирование

«Я кое-что придумал для нового продукта. Когда вы сможете это сделать?» Не отвечайте на подобный вопрос, пока больше не узнаете о проблеме. Неопределенные, плохо понятые требования порождают слишком оптимистические оценки, которые возвращаются и не дают нам покоя, когда возникает перерасход. Неподготовленная оценка звучит как обязательство для слушателя. Наибольшие вклады в проект при плохо просчитанной смете составляют затраты на частые изменения требований, пропущенные требования, недостаточное взаимодействие с пользователями, недетализированную спецификацию требований и плохой анализ (Davis, 1995). Оценка трудоемкости и продолжительности проекта на основе требований означает, что вы должны что-то знать об объеме своих требований и продуктивности команды разработчиков. Подробнее о выполнении оценки на основе требований см. раздел «Подробнее о требованиях к ПО» (Wiegers, 2006) в главе 5.

«Разрастание» требований пользователей

В процессе разработки требования могут меняться из-за чего проект часто выходит за установленные рамки как по срокам, так и по бюджету. Чтобы управлять пределами разрастания требований, для начала уточните бизнес-цели проекта, стратегическое видение, ограничения и критерии успеха. Оцените, как все предполагаемые новые характеристики или требования, отразятся на этих параметрах. Требования *будут* изменяться и расти. Менеджер проекта должен предусмотреть «буферы планирования» на случай непредвиденных обстоятельств, чтобы первое же новое требование не привело к срыву графика (Wiegers, 2007). В проектах гибкой методологии объем итерации корректируется так, чтобы вписаться в заданный бюджет и длительность итерации. При появлении новых требований они размещаются в резерве (backlog) и назначаются в будущие итерации на основе приоритета. Изменения зачастую критически важны для успеха, однако они всегда имеют цену.

Двусмысленные требования

Один из симптомов двусмысленности заключается в том, что пользователь может интерпретировать одно и то же положение по-разному. (Lawrence, 1996). Другой симптом состоит в том, что у нескольких читателей требования возникает разное понимание, что оно означает. В главе 11 указано множество слов и фраз, которые порождают двусмысленность, возлагая ответственность интерпретации на читателя.

Двусмысленность ведет и к формированию различных ожиданий у заин-

тересованных лиц. Впоследствии некоторые из них удивляются результату. Разработчики же впустую тратят время, занимаясь не теми задачами. А тестировщики готовятся к проверке не тех особенностей поведения системы.

Один из способов избавиться от двусмысленности — пригласить различных представителей пользователей для официальной экспертизы требований. (Wiegers, 2002). Как говорится в главе 17, неформальная дружественная оценка, в которой участники просто самостоятельно читают требования, часто не обнаруживает двусмысленности. Если они интерпретируют требования различными способами, но это имеет смысл для каждого из них, то неясность не проявится. Совместное выявление и проверка требований стимулирует обсуждение и уточнение требований в группе в рамках совещания. Другой способ обнаружить двусмысленность — написать вариант тестирования для требования и построить прототип.

Требования-«бантики»

Под «бантиками» (gold plating) понимают отсутствующие в спецификации требований функции, добавленные разработчиками, потому что им кажется, что это понравится пользователям. Если избыточные возможности оказываются ненужными для клиентов, получается, что время, отведенное на реализацию, тратится впустую. Прежде чем просто вставлять новые функции, разработчики и бизнес-аналитики должны представить свои творческие идеи на суд заказчиков. Задача команды — четко соблюдать требования спецификации, а не действовать за спиной клиентов, без их одобрения.

Пользователи иногда требуют функции или элементы интерфейса, которые выглядят красиво, но не представляют особой ценности для продукта. Все, что вы захотите добавить, стоит времени и денег, поэтому постарайтесь максимизировать пользу от будущего продукта. Чтобы снизить количество «бантиков», отслеживайте каждый элемент функциональности до его источника, чтобы четко понимать, почему именно он включен в продукт. Убедитесь, что все специфицируемое и разрабатываемое находится в рамках проекта.

Пропущенные классы пользователей

Большинство продуктов предназначены для нескольких групп пользователей, которые могут применять различные наборы функций с разной частотой и иметь самый разный опыт работы с ПО. Если вы не определили важные классы пользователей для вашего продукта заранее, некоторые потребности клиентов не будут учтены. После идентификации всех классов удостоверьтесь, что голос каждого услышан, как описано в главе 6. Помимо очевидных пользователей не забудьте о сотрудниках поддержки, у которых есть собственные требования, функциональные и нефункциональные. У сотрудников, которые конвертируют данные из унаследованных систем, есть требования по переходу, которые не влияют на конечный продукт, но определенно влияют на успех решения. Могут быть заинтересованные лица, которые даже

не знают о существовании проекта, например государственные учреждения, определяющие стандарты, которые могут влиять на вашу систему, и вы должны знать о таких заинтересованных лицах и их влиянии на проект.

Выгоды от высококачественного процесса разработки требований

Многие люди ошибочно считают, что время, которое тратится на обсуждение требований, просто отсрочивает выпуск продукта. Они предполагают, что работа с требованиями не повышает рентабельность проекта. На самом деле, затраты на получение хороших требований практически всегда возвращаются с излишком.

В удачных процессах создания требований к совместной партнерской работе над проектом привлекаются множество заинтересованных лиц, причем на протяжении всего проекта. Сбор требований позволяет команде разработчиков лучше понять пользователей или реалии рынка, что критически важно для любого проекта. Делая упор на задачи пользователей, а не на внешне привлекательные функции, команда избежит необходимости переписывать код, который даже не понадобится. Вовлечение клиентов в процесс снижает вероятность появления ложных ожиданий, возникающих из-за различия того, в чем пользователи нуждаются, и того, что разработчики выпускают. Рано или поздно вы начнете получать отзывы заказчиков. Причем намного дешевле добиться взаимопонимания до начала разработки продукта, чем после финальной поставки. О природе партнерства между заказчиком и разработчиками см. главу 2.

Ясное разделение требований на те, что относятся к ПО, оборудованию или подсистемам, взаимодействующим с людьми, позволяет применять системный подход к разработке продукта. Эффективные процессы управления изменениями минимизируют неблагоприятные последствия от изменения требований. Недвусмысленно составленные документы облегчают тестирование продукта. В совокупности все это повышает шансы создать высококачественный продукт, который удовлетворит всех пользователей.

Никто не станет обещать конкретный возврат инвестиций в процесс улучшения требований. Вы можете мысленно проанализировать, как более качественные требования могут помочь вашим командам (Wiegers, 2006). Стоимость более качественных требований складывается из стоимости разработки новых процедур и шаблонов документов, тренинга команды и покупки инструментов. Наибольшая инвестиция — это время, которое ваша команда тратит на сбор, документирование, просмотр и управление требованиями. Возможные выгоды таковы:

- меньше дефектов в требованиях и в готовом продукте;
- меньше переделок;
- быстрее разработка и поставка готового продукта;

- меньше ненужных и неиспользуемых функций;
- ниже стоимость модификации;
- меньше недопонимания;
- меньше расползание границ проекта;
- меньше беспорядок в проекте;
- выше удовлетворение заказчиков и членов команды;
- продукты, которые делают то, что от них ожидается.

Даже если вы не можете количественно оценить все преимущества, они все равно реальны.

Что дальше?

- Опишите связанные с требованиями проблемы, с которыми вы сталкивались в текущем или предыдущих проектах. Разбейте их на проблемы разработки или проблемы управления. Опишите основные причины каждой проблемы и ее влияние на проект.
- Организуйте обсуждение проблем с требованиями с членами вашей команды и всеми заинтересованными лицами, записывая проблемы, с которыми вы сталкивались в текущем или предыдущих проектах, их влияние и их основные причины. Поделитесь своими идеям по изменению текущих процессов работы с требованиями, которые позволят противостоять этим трудностям. Возможно, вам окажется полезным руководство по устранению неполадок в Приложении Б.
- Сравните используемую в вашей организации терминологию в области требований и продуктов и ту, что употребляется в этой главе, чтобы определить, охватываете ли вы все рекомендуемые нами категории.
- Выполните простую оценку на нескольких страницах одного из документов требований, чтобы определить области, в которых ваша команда может улучшить свою работу. Особенно полезно было бы, если бы эту оценку выполнил внешний специалист.
- Организуйте тренинг, посвященный требованиям, для команды, которая работает над вашим проектом. Пригласите основных заказчиков, маркетологов, менеджеров, разработчиков, тестировщиков и других заинтересованных лиц. Тренинг позволяет выработать единую лексику и общее понимание эффективных приемов и поведения, чтобы члены команды могли более эффективно решать стоящие перед ними общие задачи.

Глава 2

Требования с точки зрения клиента

Герхард, старший менеджер компании Contoso Pharmaceuticals, встретился с Синтией, начальником ИТ-отдела. «Нам нужно создать новую систему учета химических препаратов Chemical Tracking System, — начал Герхард. — Она должна обеспечить учет всех химических контейнеров на складе и в лабораториях. Если химикам понадобится новый реактив, который уже есть в компании, они смогут взять его в соответствующем отделе, не тратя деньги на заказ нового контейнера. Система сэкономит компании уйму денег. Кроме того, она позволит отделу охраны труда и техники безопасности тратить меньше сил на создание предоставляемых в контролирующие органы отчетов об использовании и утилизации химикатов. Ты сможешь создать систему к началу аудита, который у нас будет через пять месяцев?»

«Понимаю, почему этот проект важен, Герхард,— сказала Синтия. — Но прежде чем я набросаю график разработки проекта, нам потребуется собрать требования к системе».

Герхрад удивился: «Что вы имеете в виду? Я только что перечислил вам требования».

«На самом деле вы описали общие бизнес-цели проекта, — объяснила Синтия. — Бизнес-требования такого высокого уровня не дают достаточно информации, чтобы точно определить, какую систему создавать и сколько времени на это может потребоваться. Я хочу, чтобы мой аналитик поработал с несколькими пользователями и понял, что они ожидают от системы».

«Химики — занятые люди, — запротестовал он. — Вряд ли у них найдется время объяснять все подробности до того, как вы начнете писать программу. Не могут ли ваши люди сами определить, что создавать?»

Синтия попыталась объяснить: «Если мы сами будем пытаться угадать ожидания пользователей, ничего хорошего не выйдет. Мы — разработчики ПО, а не химики. Я по собственному опыту знаю, что, если не потратить время на изучение проблемы до начала программирования, результат не понравится никому».

«У нас нет времени на это, — настаивал Герхард. — Я описал вам мои требования. Теперь, пожалуйста, просто создайте систему. Сообщайте мне о ходе работы».

Такие диалоги регулярно возникают при разработке ПО. Клиенты, которым требуется новая информационная система, зачастую не понимают, насколько важно непосредственно опросить будущих пользователей и других заинтересованных лиц. Специалисты по маркетингу, разработавшие концепцию нового замечательного продукта, считают, что могут адекватно представлять интересы предполагаемых покупателей. Тем не менее, мнение непосредственных покупателей ПО неоценимо, и заменить его чем-либо иным нельзя. Согласно некоторым современным методикам разработки ПО, например методики гибкого программирования (agile programming), представитель заказчика должен тесно взаимодействовать с командой разработчиков. Как говорится в одной книге по гибкой разработке, «успех проекта зависит от согласованных действий клиента и программистов» (Jeffries, Anderson и Hendrickson, 2001).

Одна из проблем при формировании требований в том, что люди путают разные уровни требований, описанные в главе 1: бизнес-уровень, уровень пользователя и функциональный. Герхард перечислил несколько преимуществ, которые, по его мнению, получит компания Contoso, внедрив новую систему контроля химикатов под названием Chemical Tracking System. Однако он не знает требований пользователей, поскольку сам не работает с этой системой. С другой стороны, пользователи могут описать необходимые им возможности системы, но не способны грамотно перечислить функциональные требования, которые должны реализовать разработчики для предоставления им таких возможностей. Бизнес-аналитики должны поработать с пользователями, чтобы лучше понять будущую систему.

В этой главе речь пойдет о взаимосвязи клиента и разработчика, жизненно важной для успеха проекта по разработке ПО. Я предлагаю вашему вниманию «Билль о правах клиента ПО» и соответствующий «Билль об обязанностях клиента ПО» при формировании требований. Таким образом, я надеюсь прояснить роль клиента, а конкретнее, пользователя, в процессе создания требований. В этой главе также обсуждается критичность достижения соглашения о наборе требований, планируемых на определенный выпуск или итерацию разработки. В главе 6 описываются различные типы клиентов и пользователей и способы привлечения соответствующих представителей к выявлению требований.

Страх отказа

Недавно я побывал в отделе информационных систем одной фирмы и услышал печальную историю. Разработчики только что создали новую внутрикорпоративную систему. Пользователи с самого начала не хотели общаться с разработчиками, и когда те с гордостью представили новую систему, пользователи отвергли ее как совершенно неприемлемую. Разработчики, приложившие немало усилий, чтобы удовлетворить потребности пользователей, как они их понимали, испытали настоящий шок. И что же они предприняли? Да просто все исправили. При несоответ-

ствии требований систему всегда можно подправить, однако это значительно дороже, чем если бы пользователи оговорили свои потребности с самого начала.

Безусловно, разработчикам пришлось потратить на доводку проекта больше время и, значит, отложить следующий проект. Это абсолютно проигрышная ситуация. Разработчики растеряны и расстроены, клиенты разочарованы, так как система не оправдала их ожиданий, а компания потеряла кучу денег и возможностей, которые предоставили бы другие проекты, которые пришлось отложить. Когда клиенты с самого начала плотно и постоянно вовлечены в разработку системы, такого неудачного, но, к сожалению, нередкого итога удалось бы избежать.

Разрыв ожиданий

Без адекватного участия клиента в конце проекта неизбежно возникает *разрыв ожиданий* (expectation gap) — пробел между тем, что клиенту реально нужно, и тем, что предоставили разработчики, основываясь на том, что они знали в начале проекта (Wiegers, 1996). Это показано пунктирными линиями на рис. 2-1. Разрыв ожиданий становится неприятной неожиданностью для всех заинтересованных лиц. По собственному опыту могу сказать, что сюрпризы в ПО никогда не бывают хорошими. Требования устаревают по мере изменения в бизнесе, поэтому взаимодействие с клиентами жизненно важно.

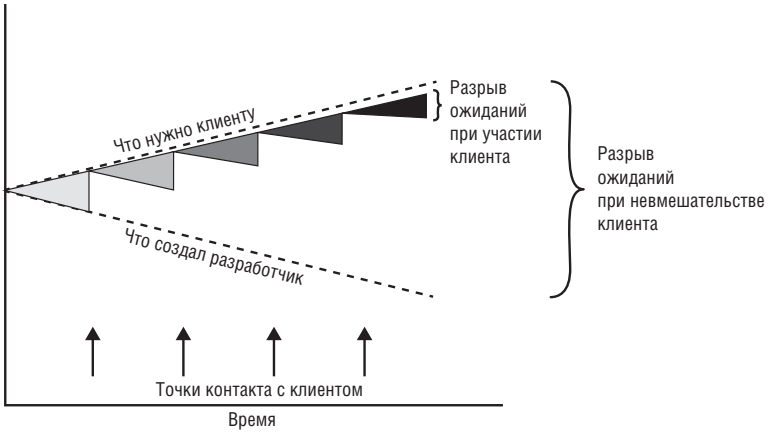


Рис. 2-1. Частое взаимодействие с клиентом сокращает разрыв ожиданий

Наилучший способ минимизации разрыва ожиданий — организация частых точек контакта с представителями клиента. Эти точки могут принимать форму интервью, собеседований, просмотра требований, просмотра дизайна пользовательского интерфейса, оценки прототипа, а в случае гибкой (agile) разработки — откликов клиентов на небольшие расширения функциональности исполняемого ПО. Каждая точка контакта предоставляет возможность

закрыть разрыв ожиданий: создаваемое разработчиком ближе к тому, что требуется клиенту.

Естественно, что разрыв начинает снова увеличиваться сразу же после каждой точки контакта. Чем чаще такие точки, тем проще двигаться в правильном направлении. Как иллюстрируют маленькие серые треугольники на рис. 2-1, набор таких точек контакта позволяют в конце проекта получить намного меньший разрыв ожиданий и решение, которое намного лучше соответствует реальным потребностям клиента. Вот почему основным принципом гибкой разработки является постоянный диалог между разработчиками и клиентами. Этот принцип отлично подходит для любого проекта.

Кто же клиент?

Прежде чем говорить о клиентах, нужно обсудить заинтересованных лиц. *Заинтересованное лицо* (stakeholder) — это человек, группа или организация, которая активно задействована в проекте, подвержена влиянию процесса или результата или может влиять на процесс или результат. Заинтересованные лица могут быть внутренними или внешними по отношению к команде проекта или разрабатывающей организации. На рис. 2-2 показаны возможные заинтересованные лица в этих категориях. Естественно, что не в каждом проекте присутствуют все эти лица.

Рис. 2-2. Возможные заинтересованные лица в команде проекта, в разрабатывающей организации и за пределами организации

Анализ заинтересованных лиц — важная часть разработки требований (Smith, 2000; Wiegers, 2007; IIBA, 2009). При определении возможных заинтересованных лиц конкретного проекта, надо смотреть максимально широко, чтобы не пропустить важных участников. После этого можно основательно проанализировать список возможных заинтересованных лиц, чтобы определить чье участие вам действительно нужно для понимания всех требований и ограничений проекта, которые позволят команде предоставить правильное решение.

Клиенты являются подмножеством заинтересованных лиц. *Клиент* (customer) — человек или организация, получающая от продукта прямую или косвенную выгоду. Клиенты это заинтересованные в проекте лица, запрашивающие, оплачивающие, выбирающие, определяющие, использующие и получающие результаты работы программного продукта. Показанные на рис. 2-2 клиенты включают непосредственного пользователя, непрямого пользователя, куратора, специалистов по закупкам и покупателя. Некоторые заинтересованные лица не являются клиентами, такие как юристы, аудиторы, поставщики, подрядчики и венчурные инвесторы. Менеджер Герхард — это клиент, оплачивающий или курирующий проект по разработке ПО. Клиенты уровня Герхарда определяют бизнес-требования к системе. Они формулируют высокоуровневую концепцию продукта и бизнес-обоснование для его развертывания. Как вы узнаете из главы 5, бизнес-требования описывают бизнес-цели, которых хочет достичь клиент, компания или другие заинтересованные лица. Любые другие функции и требования к продукту должны удовлетворять бизнес-требованиям.

Требования пользователей определяют те, кто прямо или косвенно взаимодействуют с продуктом. Эти пользователи (часто их называют *конечными пользователями*) являются подмножеством клиентов. Прямые пользователи непосредственно работают с продуктом. Непрямые пользователи могут получать результаты работы системы, не входя в непосредственный контакт с ней, например менеджер хранилища данных, получающий по почте ежедневный отчет о деятельности хранилища данных. Пользователи способны описать, какие задачи им нужно выполнять, какие результаты они ожидают и какие ожидаются качественные характеристики продукта.

Случай с отсутствующим заинтересованным лицом

Я знаю проект, в котором на финальном этапе выявления требований при проверке правильности потока процессов аналитик поинтересовался у заинтересованного лица: «Вы уверены в правильности шагов вычисления налога в этом потоке?» На что получил такой ответ: «Ну, я не знаю. Я не занимаюсь налогами — для этого есть отдел по налогам». На протяжении многих месяцев работы над проектом никто из команды не разговаривал ни с одним сотрудником отдела по налогам. В команде даже не догадывались, что такой отдел *вообще существует*. При встрече с сотрудниками

отдела по налогам аналитик обнаружил длинный список пропущенных требований, связанных с несоответствием реализации налоговой функциональности и требований регулирующих органов. В результате проект был задержан на несколько месяцев. Подобных неприятностей можно избежать, если заранее ознакомиться со структурной схемой организации и выявить всех заинтересованных лиц, на работу которых может повлиять новая система.

Клиенты, предоставляющие бизнес-требования, иногда пытаются говорить от имени реальных пользователей, но обычно они слишком далеки от реальной работы, чтобы описать их точные требования. В случае с информационными системами или разработкой нестандартного приложения бизнес-требования должен определять тот, кто в конечном итоге отвечает за пользу для бизнеса, ожидаемую от продукта. Пользовательские требования должны определять те, кто будет стучать по клавишам, касаться экрана и получать результаты работы продукта. В случае серьезного рассогласования между заказчиками, оплачивающими проект, и конечными пользователями крупных проблем не избежать.

В области разработки коммерческого ПО, где клиент и пользователь зачастую представлены одним лицом, ситуация несколько иная. Представители клиента, например маркетологи или менеджеры по продукту, обычно пытаются на свой вкус определить, что клиент счел бы привлекательным. Тем не менее, без конечных пользователей сформулировать требования пользователей не удастся (подробнее — в главе 7). В противном случае будьте готовы читать обзоры в журналах, описывающие недостатки вашего продукта, которых удалось бы избежать при активном участии пользователей.

Возможны конфликты между заинтересованными лицами проекта. Бизнес-требования иногда отражают организационную стратегию или бюджетные ограничения, которые не всегда понятны пользователям. И те, раздраженные тем, что менеджмент насильно внедряет новую информационную систему, не всегда хотят общаться с разработчиками ПО, считая последних предвестниками неблагоприятного будущего. Таких пользователей иногда называют «группами неудачников» (Gause и Weinberg, 1989). Избежать таких трений помогает ясное и полное обсуждение целей и ограничений проекта, которое может убедить пользователей и избежать споров и обид.

Сотрудничество клиентов и разработчиков

Отличные программные продукты — результат правильно выполненного проектирования, основанного на требованиях, полученных в результате эффективного, то есть тесного взаимодействия разработчиков и клиентов (и особенно, фактических пользователей). Совместная работа возможна только тогда, когда все участники процесса разработки знают, что именно необходимо им для успеха, и когда они понимают и уважают стремление их со-

ратников к успеху. Когда при работе над проектом нагнетается напряжение, очень легко забыть, что у всех участников единая цель — создать программный продукт, ценный для бизнеса и отвечающий чаяниям всех участников проекта. Аналитик обычно оказывается ключевым человеком, на плечи которого ложится налаживание этого сотрудничества.

«Билль о правах клиента ПО» (табл. 2-1) содержит 10 положений, на выполнении которых клиенты могут на вполне законных основаниях настаивать при общении с аналитиками и разработчиками на этапе формулирования требований к проекту. Каждый пункт этого права описывает соответствующую ответственность аналитиков и разработчиков ПО. Слово «вы» в правах и обязанностях указывает на клиента в проекте разработки ПО.

Табл. 2-1. Билль о правах клиента ПО при формировании требований

У вас есть право
1. Иметь дело с аналитиком, который разговаривает на вашем языке
2. Иметь дело с аналитиком, хорошо изучившим ваш бизнес и цели, для которых создается система
3. Потребовать, чтобы аналитик зафиксировал требования в надлежащей форме
4. Получить подробный отчет о будущих процедурах и результатах процесса формулирования требований
5. На изменение ваших требований
6. На взаимное уважение
7. Знать о вариантах и альтернативах требований и их реализации
8. Описать характеристики, упрощающие работу с продуктом
9. Узнать о способах корректировки требований для ускорения разработки за счет повторного использования
10. Получить систему, функциональность и качество которой соответствует вашим ожиданиям

Так как обратной стороной прав являются обязанности, «Билль об обязанностях клиента ПО» (табл. 2-2), напротив, содержит 10 положений, определяющих ответственность клиента перед аналитиком и разработчиком на этапе формулирования требований. Возможно, его стоит назвать «Билль о правах разработчика». Если эти документы не совсем точно подходят для вашей организации, измените их в соответствии с реалиями местной специфики.

Перечисленные права и обязанности распространяются непосредственно на клиентов в случае, когда программный продукт разрабатывается для внутрикорпоративного использования, на заказ или для определенной группы крупных клиентов. При разработке продуктов для массового рынка интересы клиентов представляют сотрудники, например отдел маркетинга.

Табл. 2-2. Билль об обязанностях клиента ПО при формировании требований

Клиент обязан
1. Ознакомить аналитиков и разработчиков с особенностями вашего бизнеса
2. Потратить столько времени, сколько необходимо на уточнение требований
3. Точно и конкретно описать требования к системе
4. Принимать своевременные решения относительно требований
5. Уважать определенную разработчиком оценку стоимости и возможности реализации ваших требований
6. Определять реалистичные приоритеты требований совместно с разработчиками
7. Проверять требования и оценивать прототипы
8. Определить критерии приемки
9. Своевременно сообщать об изменениях требований
10. Уважительно относиться к процессам создания требований

В процессе планирования проекта клиенту и разработчикам следует изучить оба этих списка и постараться достигнуть взаимопонимания. Убедитесь, что основные участники процесса формулирования требований понимают и принимают свои обязанности. Это позволит избежать трений позже, когда одна сторона будет ожидать чего-то, что другая не может или не желает предоставить.

> **Внимание!** Не предполагайте, что участники проекта интуитивно знают, как сотрудничать при формулировании требований. Потратьте время и обсудите, каким образом организовать взаимодействие наиболее эффективно. Хорошо записать, как вы хотите решать и управлять вопросами требований в проекте. Это послужит ценным средством коммуникации в проекте.

Билль о правах клиента ПО

Далее перечислены десять прав, наличие которых предполагают клиенты, когда речь идет о вопросах требований.

Право № 1. Иметь дело с аналитиком, который разговаривает на вашем языке

При обсуждении требований следует выяснить потребности и задачи вашего бизнеса, используя при этом принятую в вашем бизнесе терминологию. Заставьте аналитиков говорить с вами на вашем языке (возможно, для этого им следует предоставить небольшой словарь). При разговоре с аналитиками не надо переходить на технический жаргон.

Право № 2. Иметь дело с аналитиком, хорошо изучившим ваш бизнес и цели, для которых создается система

Выявляя требования, аналитики смогут лучше понять задачи вашего бизнеса и осознать, какое место уготовано создаваемому ПО в вашем бизнесе. Это поможет им удовлетворить ваши ожидания. Пригласите разработчиков и аналитиков посмотреть на вашу работу. Если заказанная система заменит существующее приложение, предложите аналитикам поработать с ним. Таким образом, им будет легче понять его сильные и слабые стороны и то, как его можно усовершенствовать. Не надо предполагать, что аналитик уже знает все ваши бизнес-операции и терминологию (см. Обязанность №1).

Право № 3. Потребовать, чтобы аналитик зафиксировал требования в надлежащей форме

Аналитик разберется со всей информацией, предоставленной заинтересованными лицами, и задаст дополнительные вопросы, чтобы пользовательские требования на основе бизнес-требований, бизнес-правил, функциональных требований, целей качества и прочих элементов. Конечный итог этого анализа — выверенный набор требований, хранящийся в одной из надлежащих форм, например спецификация требований к программному обеспечению (SRS) или средство управления требованиями. Этот набор требований является соглашением заинтересованных лиц относительно функций, качеств и ограничений продукта, который планируется построить. Требования должны быть написаны и организованы так, чтобы их было легко понимать. Ваша проверка этих спецификаций и других представлений требований, например моделей визуального анализа, помогает обеспечить точное представление ваших потребностей.

Право № 4. Получить подробный отчет о будущих процедурах и результатах процесса формулирования требований

Различные процедуры могут сделать процесс формулирования и управления требованиями эффективным и продуктивным, а знание требований может представляться в самых разных формах. Аналитик должен объяснить рекомендуемые процедуры, а также определить какую информацию будет содержать каждый полученный в результате документ. Например, для иллюстрации текста аналитик может создать ряд диаграмм. Эти диаграммы могут показаться непривычными или сложными, но они просты. Аналитик должен объяснить назначение каждой из них, смысл обозначений и процедуру проверки диаграмм на предмет ошибок. Если он не предоставляет таких объяснений, попросите его представить их.

Право № 5. Изменить свои требования

Со стороны аналитиков или разработчиков нереалистично ожидать, что вы сразу сможете предоставить все требования или что эти требования останут-

ся неизменными на протяжении всего цикла разработки. У вас есть право вносить изменения в требования по мере развития бизнеса, в процессе получения членами команды больше информации от заинтересованных лиц или когда вы лучше обдумаете, что на самом деле вам нужно. Тем не менее, у изменений всегда есть цена. Иногда при добавлении новой функции приходится приносить в жертву другие функции или вносить коррективы в график или бюджет проекта. Важная часть ответственности аналитика заключается в оценке, управлении или изменении последствий подобных действий. Совместно с аналитиком выработайте простой, но эффективный процесс работы с изменениями в своем проекте.

Право № 6. На взаимное уважение

Отношения между клиентами и разработчиками могут становиться конфликтными. Если клиенты и разработчики не понимают друг друга, обсуждение требований может обернуться большим разочарованием. Совместная работа позволяет открыть друг другу глаза на проблемы, с которыми сталкивается каждая из этих групп. Клиенты, участвующие в процессе выработки требований, имеют полное право требовать от аналитиков и разработчиков уважительного отношения к себе и бережного отношения к затраченному времени. В свою очередь, и клиентам следует оказывать уважение разработчикам, ведь они все вместе сотрудничают для достижения общей цели — успешного проекта. Все находятся по одну сторону баррикад.

Право № 7. Знать о вариантах и альтернативах требований и их решении

Чтобы гарантировать, что новая система не будет автоматизировать неэффективные или устаревшие процессы, аналитик должен знать, почему существующие системы не годятся для ваших бизнес-процессов. Аналитики часто предлагают те или иные усовершенствования ваших бизнес-процессов. Полезен и аналитик, творчески подходящий к делу: он предлагает новые возможности программы, о которых пользователи даже и не мечтали.

Право № 8. Описать характеристики, упрощающие работу с продуктом

Вполне вероятно, что аналитики спросят вас о характеристиках ПО, выходящих за рамки функциональных потребностей пользователей. Благодаря им программный продукт становится более удобным в работе, что позволяет клиентам эффективнее выполнять свои задачи. Иногда пользователи просят, чтобы продукт был *дружественным* или *надежным*, однако эти термины слишком субъективны, чтобы помочь разработчикам. Поэтому аналитик должен выяснить конкретные характеристики, означающие для вас дружественность или надежность. Расскажите аналитику о том, какие стороны ваших текущих приложений кажутся вам «дружественными», а какие нет. Если вы

не обсудите эти характеристики с аналитиком, вам должно сильно повезти, чтобы продукт получился именно таким, как вы надеялись.

Право № 9. Узнать о способах корректировки требований для ускорения разработки за счет повторного использования

Отношение к требованиям должны быть гибкими. Аналитику могут быть известны готовые программные компоненты или требования, которые практически полностью удовлетворят некоторые названные вами потребности. В таком случае аналитику следует предложить скорректировать ваши требования, чтобы разработчики могли использовать имеющееся ПО. Разумные возможности применения существующих модулей сэкономит ваши время и деньги. Если вы посчитаете разумным включить в свой продукт готовые коммерческие компоненты, будьте готовы проявить гибкость, поскольку характеристики таких компонентов вряд ли будут точно соответствовать вашим потребностям.

Право № 10. Получить систему, функциональность и качество которой соответствует вашим ожиданиям

Это самое главное право клиента, *но* оно осуществимо, только если вы сумеете донести до разработчиков всю информацию, которая поможет им создать подходящий продукт, если разработчики четко изложат вам все варианты и ограничения и если все участники достигнут соглашения. Убедитесь, что вы высказали все свои ожидания и предположения; в противном случае программисты, скорее всего, не смогут реализовать их. Иногда клиенты не формулируют определенные вещи, считая их общеизвестными фактами. Однако проверка правильности понимания всеми членами команды так же важна, как выражение чего-то нового.

Билль об обязанностях клиента ПО

Не бывает прав без обязанностей, поэтому приводим следующие десять обязанностей представителя клиента в процессе определения и управления требованиями в проекте.

Обязанность №1. Ознакомить аналитиков и разработчиков с особенностями вашего бизнеса

Только вы можете полноценно познакомить разработчиков с концепциями и терминологией своего бизнеса. Вам не надо делать аналитиков экспертами в предметной области, основная задача — помочь им понять ваши проблемы и цели. Не ожидайте, что аналитики постигнут нюансы и неявные особенности бизнеса. Скорее всего, у них нет знаний, воспринимаемых вами и вашими коллегами, как должное.

Обязанность №2. Потратить столько времени, сколько необходимо на предоставление и уточнение требований

Клиенты — занятые люди, и те, кто участвует в формулировании требований — обычно самые занятые из них. Тем не менее, вы обязаны потратить время на участие в совещаниях, мозговых штурмах, интервью и прочих процедурах, необходимых для выявления требований. Иногда аналитик считает, что понял вашу идею, а позже сообразит, что ему необходимы дополнительные разъяснения. Пожалуйста, терпеливо относитесь к такому поэтапному подходу к формулированию и прояснению требований; это — природа сложного человеческого общения и ключ к успеху создаваемого ПО. Общее затраченное время будет меньшим, если выделить несколько часов на целенаправленное формулирование требований, чем растягивать процесс на долгие недели, вырывая минутку там и здесь на работу с требованиями.

Обязанность №3. Точно и конкретно описать требования к системе

Весьма заманчиво оставить требования неопределенными и нечеткими, ведь прояснять подробности так утомительно и долго. Тем не менее, на каком-то этапе разработки участникам проекта необходимо устранить все неоднозначности и неточности. Вам, как клиенту, и карты в руки. В противном случае угадывать, что же вам именно нужно, будут аналитики или разработчики. В спецификацию требований к программному обеспечению рекомендуется включить маркеры «требует прояснения» (to be determined, TBD), указывающие на необходимость дополнительных исследований, анализа и информации. Однако иногда такие маркеры используют в случаях, когда конкретное требование трудно определить точно и никто не хочет с ним возиться. Попытайтесь прояснить цель каждого требования, чтобы аналитик мог точно выразить его. Это самый лучший способ гарантировать, что продукт будет отвечать вашим потребностям.

Обязанность №4. По запросу принимать своевременные решения относительно требований

Точно так же, как и подрядчик при строительстве дома, аналитик задаст вам множество вопросов и попросит выбрать различные варианты и принять решения. Это может быть согласование противоречивых запросов от разных клиентов, выбор между конфликтующими атрибутами качества и оценка точности информации. Клиенты, обладающие соответствующими полномочиями, должны принять эти решения, когда их об этом попросят. Зачастую работа стопорится, так как клиент не может принять окончательного решения, и поэтому, медля с ответом, вы затягиваете работу над проектом. Когда потребности в вашем времени становятся обременительными, помните, что система создается для вас. Аналитики обучены помогать людям принимать решения, поэтому обратитесь к ним за помощью, когда не можете решиться на что-то.

Обязанность №5. Уважать определенную разработчиком оценку стоимости и возможности реализации ваших требований

Все программные функции чего-то стоят. Разработчики лучше всего способны оценить эту стоимость. Может оказаться, что некоторые необходимые вам функции технически неосуществимы или их реализация слишком дорога, например недостижимая в рабочей среде производительность или доступ к данным, которые системе просто недоступны. Даже если сведения об осуществимости и стоимости некоторых функций, сообщенные разработчиком, вам не понравятся, следует уважать эту оценку. Иногда удается переписать требования так, чтобы они стали реализуемыми или стоили бы меньше денег. Например, требовать «моментального» выполнения операции неразумно, а более точное требование («в течение 50 миллисекунд») может оказаться вполне реализуемым.

Обязанность №6. Определять реалистичные приоритеты требований совместно с разработчиками

Лишь при работе над ограниченным кругом задач можно рассчитать время и ресурсы для реализации всей желаемой функциональности. Определить, какие возможности необходимы, какие полезны и без каких пользователи обойдутся — важная составляющая анализа требований. Именно вы определяете эти приоритеты, поскольку разработчики не в состоянии это сделать. Чтобы вам облегчить задачу, разработчики предоставляют информацию о стоимости и рисках всех требований. Определив приоритеты, вы поможете разработчикам вовремя и с минимальными затратами создать максимально эффективный продукт. Совместная приоритизация является ключевой в проектах гибкой (agile) разработки — она позволяет разработчикам максимально быстро поставлять полезный программный продукт.

Уважайте оценку команды разработчиков, какой объем функциональности они могут завершить за отведенное время и с имеющимися ограничениями по ресурсам. Если все необходимое не умещается в рамки проекта, ответственные за принятие решений лица должны сократить объем задач, основываясь на приоритетах, увеличить длительность проекта или предоставить дополнительный бюджет или людей. Просто объявить все требования высокоприоритетными нереалистично и не соответствует духу сотрудничества.

Обязанность №7. Проверять требования и оценивать прототипы

Как говорится в главе 15, рецензирование требований — одна из наиболее значимых операций, обеспечивающих качество ПО. Участие клиентов в таких мероприятиях — единственный способ узнать, отражают ли требования потребности клиента наиболее полно и точно. Обзор также является возможностью для представителей клиента оценить, насколько хорошо работа аналитика соответствует потребностям проекта. Занятые клиенты часто не хотят тратить время на обзор требований, но это стоит того. Аналитик должен предоставлять требования для обзора разумными порциями на всем протя-

жении процесса выявления требований, а не сваливать на ваш стол объемную пачку документации, когда требования «готовы».

Читая спецификацию требований, не всегда удается четко представить работу программы. Чтобы лучше понять потребности клиента и выявить оптимальные способы их удовлетворения, аналитики и разработчики иногда создают прототипы предполагаемого продукта. Ваши отклики на эти предварительные, частичные или пробные версии дают разработчикам необходимую информацию.

Обязанность №8. Определить критерии приемки

Как разработчики определят, что выполнили свою задачу? Как им узнать, соответствует ли созданное ПО ожиданиям различных сообществ клиентов? Одна из обязанностей клиента — установить критерии приемки, то есть заданные условия, которым должен удовлетворять продукт, чтобы его можно было считать приемлемым. Такие критерии включают приемочные тесты, которые оценивают, позволяет ли продукт пользователям правильно выполнять те или иные их операции. Другие критерии приемки могут определять, допустимый уровень оставшихся в продукте дефектов, производительность определенных действий в рабочей среде или способность удовлетворить определенным внешним сертификационным требованиям. В проектах гибкой разработки (agile) вместо письменных требований активно используются приемочные тесты для наполнения «плотью и кровью» пользовательских историй. Тестировщики могут определить, правильно ли реализовано то или иное требование, но они не всегда знают, что *вы* считаете допустимым результатом.

Обязанность №9. Своевременно сообщать об изменениях требований

Постоянное изменение требований — серьезная угроза для возможности своевременной сдачи проекта командой разработчиков. Изменение неизбежно, но чем позже в ходе разработки о нем сообщается, тем более сильное влияние оно окажет. Как только вы решите изменить требования, сразу же сообщите об этом аналитику, с которым работаете. Для снижения негативного влияния изменений следуйте определенному в проекте процессу управления изменениями. Это гарантирует, что запрошенные изменения не потеряются, влияние каждого изменения будет проанализировано и все предложенные изменения будут рассмотрены с соблюдением единой процедуры. В результате заинтересованные лица компании смогут принять разумные бизнес-решения по внедрению соответствующих изменений на правильном этапе проекта.

Обязанность №10. Уважительно относиться к процессам создания требований

Выявление и спецификация требований — одни из самых трудных задач в разработке ПО. У всех подходов, применяемых аналитиками, есть рацио-

нальная основа. И хотя операции по созданию требований могут вас разочаровать, время, затраченное, чтобы разобраться в требованиях, — это отличные инвестиции. Процесс окажется менее болезненным, если вы разберетесь в методах, используемых аналитиками для создания требований. Не стесняйтесь и просите аналитиков объяснить, зачем необходимы те или иные сведения и почему они просят вас поучаствовать в некоторых операциях по созданию требований. Взаимопонимание и уважение приемов и потребностей друг друга имеют огромное значение для организации эффективного и даже доставляющего удовольствие сотрудничества.

Создание культуры уважения к требованиям

Руководитель отдела работы с требованиями однажды сформулировал проблему: «У меня сложность с получением согласия разработчиков на участие в разработке требований. Как мне помочь им понять ценность их участия?» В другой организации аналитик столкнулся с конфликтом между разработчиками, пытающимися получить подробные сведения о бухгалтерской системе, и ИТ-менеджером, который просто хотел провести мозговой штурм по определению требований вместо использования специальных приемов выявления требований. Этот аналитик спросил меня: «Решаются ли читатели вашей книги на конфронтацию культур?»

Эти вопросы отображают проблемы, которые возникают при попытке привлечь аналитиков, разработчиков и клиентов к коллективному партнерству при работе над требованиями. Можно посчитать очевидным, что предоставление пользователем информации о требованиях повышает вероятность, что он получить то, что ему нужно. Разработчики должны понять, что участие в этом процессе сделает их жизнь проще, и не придется удивляться документам требований, которые приходят из-за воображаемой стены. Естественно, не всем нравятся требования, как вам, в противном случае они бы все стали аналитиками!

При работе над требованиями часто возникают конфликты культур. Есть люди, которые понимают риски, связанные с попытками разработать ПО на основе минимальных или «телепатически» передаваемых требований. Есть также те, что считают требования ненужными. Бывает тяжело добиться сотрудничества со стороны бизнес-подразделений в проектах по замене старых систем, если пользователи видят, что это не связано с их собственными бизнес-задачами и не стоит их времени. Понимание, почему пользователи сопротивляются участию в разработке требований, — первый шаг на пути к решению проблемы.

Возможно, что противники требований никогда не работали в коллективе с прочными традициями работы с требованиями. Или они стали жертвами неудачной реализации процессов работы с требованиями, например у них был опыт работы в проекте, где создавались огромные, неполные и неиспользуемые спецификации требований. Это у любого испортит отношение к тре-

бованиям. Возможно, противники не понимают и не осознают ценность процедур работы с требованиями, если они выполняются эффективно. Они могут не осознавать цену, которую пришлось заплатить при работе в случайной и неструктурированной среде в прошлом. Цена обычно выражается в неожиданных переделках, которые приводят к задержкам выпуска и низкому качеству ПО. Подобные переделки скрыты в ежедневной работе участников проекта, поэтому они не считают их серьезной проблемой с эффективностью.

Пытаясь привлечь на свою сторону разработчиков, менеджеров и клиентов, позаботьтесь, чтобы все поняли, сквозь какие неприятности пришлось пройти организации и ее клиентам из-за проблем с требованиями. Представьте конкретные примеры, если люди не испытывали неприятности лично. Выразите затраты в единицах, понятных в организации, — это могут быть деньги, недовольство клиентов или потерянные бизнес-возможности. Менеджеры разработки не всегда понимают, насколько плохо недостатки требований сказываются на производительности их команд. Поэтому покажите им, как плохие требования замедляют проектирование и ведут к излишним и недешевым корректировкам курса.

Разработчики являются заинтересованными лицами в проекте, но бывает так, что их никто не спрашивает и они становятся «жертвами» спущенных к ним сверху требований. Поэтому им выгодно участвовать, чтобы их вклад позволил сделать документацию по требованиям максимально полезной и показательной. Мне нравится, когда разработчики просматривают требования по мере их создания. Таким образом они знают, чего ожидать, и могут указать области, в которых им нужна большая ясность. Участие разработчика также требуется при определении внутренних атрибутов качества, которые невидимы для пользователей. Разработчики могут предоставить предложения, о которых другие как правило не думают: более простые способы выполнения определенных вещей, функциональность, реализация которой может занять очень много времени, ненужные ограничения на дизайн, отсутствующие требования, например как обрабатывать исключения, а также продуктивные возможности использования технологий.

Вклад сотрудников отделов контроля качества и тестирования также важен для создания отличных требований. Не стоит откладывать на более поздние этапы привлечение этих наблюдательных людей к итеративной проверке требований. При разработке своих тестовых случаев и сценариев на основе требований они скорее всего обнаружат много неясностей, противоречий и проблем с требованиями. Тестировщики могут также предоставить информацию об определении поддающихся проверке требований к атрибутам качества.

Причиной сопротивления изменениям процессов или культуры может быть страх, неуверенность или недостаток знаний. Если вы сумеете определить источник сопротивления, у вас появится возможность бороться с ним путем увещеваний, объяснений и просвещения. Покажите людям, что их участие принесет пользу не только им лично, но от этого выиграют все.

Руководство организации должно понимать необходимость иметь эффективные возможности бизнес-анализа и разработки требований как стратегические базовые корпоративные знания. Несмотря на то, что проектная и локализационная деятельность важны, без решимости руководства преимущества и выигрыш скорее всего будут утеряны по завершении проекта или после реорганизации.

Определение ответственных за принятие решений

В процессе реализации проектов по разработке ПО может потребоваться принимать сотни решений, причем часто они находятся на критическом пути и без них невозможно двигаться вперед. Может возникнуть необходимость уладить конфликт, принять (или отвергнуть) предложенное изменение или одобрить набор требований для определенного выпуска. На ранних этапах проекта определите лиц ответственных за принятие решений в области требований и то, как они должны принимать решения. Мой друг Крис, опытный менеджер проектов, сказал: «Я обнаружил, что обычно есть один основной человек, принимающий решения в проекте, и очень часто это ключевой «куратор» в организации. Я обязательно нахожу этого человека, после чего слежу за тем, чтобы он всегда был в курсе продвижения проекта». Нет универсального правильного ответа на вопрос, кто должен принимать ключевые решения в проекте. Но лучше всего это получается у небольшой группы, представляющей самые важные области — руководство, клиентов, аналитиков, разработчиков и маркетинг. В главе 28 рассказывается о совете по изменениям, который служит органом принятия решений по предложенным изменениям требований.

В группе принятия решений нужно определить *главного ответственного* за принятие решений и *правило* принятия решений, которой описывает порядок принятия решения в группе. Существует много правил принятия решений, в том числе следующие (Gottesdiener, 2001):

- Решения принимает главный ответственный *с* или *без* предварительного обсуждения с другими.
- Члены группы голосуют и решение принимается большинством голосов.
- Члены группы голосуют, но решение принимается только единогласно.
- Члены группы обсуждают и договариваются, достигая консенсуса. Всех устраивает решение и каждый обязуется поддерживать его.
- Главный ответственный делегирует право принятия решения какому-то человеку.
- Решение принимает группа, но у одного из ее членов есть право вето на принятие решения.

Нет одного самого правильного и подходящего всем правила принятия решений. Одно правило принятия решений не может работать во всех си-

туациях, поэтому в группе нужно уставить соответствующие руководящие положения, чтобы членам было понятно, когда голосовать, когда нужен консенсус, когда делегировать и т. п. Люди, ответственные за принятие решений по требованиям, в любом из ваших проектов должны выбрать правило принятия решений до того, как начнут принимать свое первое значительное решение.

Достижение соглашения о требованиях

Достижение соглашения о требованиях к продукту или его части, которую планируется построить, — основа сотрудничества клиентов и разработчиков. В выработке соглашения участвует много сторон:

- клиенты должны подтвердить, что требования удовлетворяют их потребности;
- разработчики подтверждают, что они понимают требования и в состоянии их реализовать;
- тестировщики подтверждают, что требования поддаются проверке;
- руководство подтверждает, что требования соответствуют их бизнес-целям.

Многие организации просят клиента поставить свою подпись на документах с требованиями: это означает, что клиент их подтверждает. Все участники процесса утверждения требований должны четко понимать, что означает такая подпись и какие проблемы она может создавать. Одна из таких проблем заключается в том, что представитель или менеджер клиента может посчитать эту процедуру бессмысленной: «Мне дали лист бумаги с моим именем, и я на этой бумаге расписался, иначе разработчики не начали бы программировать». Если такой заказчик захочет через некоторое время изменить требования или его не устроит результат работы, детским лепетом прозвучат слова: «Ну да, я подписал эти требования, но у меня не было времени их читать. Я доверял вам, ребята, а вы меня подвели!»

Иногда проблему создает менеджер по разработке, если он рассматривает подпись как способ заморозить требования, сделав их неизменными. Когда клиент просит о каких-то изменениях, он будет протестовать: «Но вы же подписали эти требования, и именно такую систему мы и создаем. Если вам нужно было что-то другое, следовало сказать об этом раньше».

Оба описанных подхода игнорируют реальность, которая такова, что на ранних этапах работы над проектом нельзя знать абсолютно всех требований и со временем они неизбежно меняются. Утверждение требований — операция, которая закрывает прения на определенном этапе создания требований. Тем не менее, участникам необходимо подтвердить свои слова подписями.

> **Внимание!** Не используйте подпись в качестве оружия. Применяйте ее как завершение этапа проекта и выработайте четкое коллективное понимание того, как подпись повлияет на возможные в будущем изменения. Если ответственные за принятие решений лица не должны читать каждое слово требований, выберите средство коммуникации, например презентацию слайдов, которое предоставит сводную информацию об основных элементах и позволит быстро достичь соглашения.

Базовое соглашение о требованиях

Гораздо важнее ритуала подписи концепция создания *базового* (baseline) соглашения о требованиях — снимка такого соглашения на какой-то момент времени (Wiegers, 2006). Базовое соглашение о требованиях является набором проверенных и согласованных требований, которые служат основой для дальнейшей разработки. Использует ваша команда формальный процесс подписи или другие средства достижения соглашения о требованиях, текст под подписью на странице подтверждения требований должен звучать примерно так:

«Подтверждаю, что настоящий набор требований наилучшим образом представляет наше понимание требований к этому проекту на данный момент и что описанная система удовлетворит наши потребности. Я согласен вносить в будущем изменения в это базовое соглашение в соответствии с процессом изменения требований, определенным в проекте. Я понимаю, что изменения могут потребовать переоценки стоимости, ресурсов и сроков сдачи проекта».

В некоторых организациях подобный текст размещают прямо на странице подписи, чтобы лица, одобряющие документ, точно знали, что означает их подпись.

Согласованное понимание значения подписи позволяет избежать трений, возникающих при изменении взглядов на требования, а также при изменении рыночных и бизнес-требований к проекту. Осмысленный процесс создания базового соглашения о требованиях дает уверенность всем участникам проекта:

- руководство или отдел маркетинга клиента уверены, что границы проекта не выйдут из-под контроля, поскольку решения относительно границ принимает клиент;
- представители клиента уверены, что разработчики будут взаимодействовать с ним для создания нужной системы, даже если перед началом работы над проектом они не успели продумать все требования;
- руководство проекта уверено, что команда разработчиков получила достойного делового партнера, который ориентирован на достижение поставленных целей и готов сотрудничать с разработчиками над достижением баланса затрат, функциональности и качества;
- бизнес-аналитики и менеджеры проекта уверены в том, что могут управлять изменениями проекта, минимизируя хаос;
- команды контроля качества и тестирования могут спокойно разрабатывать свои сценарии тестирования, будучи полностью готовыми к выполнению своих обязанностей в проекте.

После определения базового соглашения о требованиях аналитик должен разместить требования в системе управления версиями. Это позволит команде при необходимости контролируемым образом изменять границы требова-

ний, включая анализ влияния каждого предлагаемого изменения на график и другие факторы успеха. Скрепив начальные операции по формулированию требований явным соглашением, вы сплотите клиентов и разработчиков на пути к успеху проекта.

Что если не удается достичь соглашения?

Может быть сложным получить подпись всех нужных заинтересованных лиц. К возможным препятствиям на пути к решению этих задач относятся логистика, напряженный график или люди, которые не хотят брать на себя обязательства, за которые потом придется отвечать. Если заинтересованные лица боятся, что не смогут внести изменения после одобрения требований, они будут тянуть с одобрением. Это одно из обстоятельств, способных привести к ужасно нежелательной ситуации паралича анализа. Во многих командах пытаются отправить примерно такое сообщение электронной почты: «Если вы не представите ответ со своими коррективами или подписью до следующей пятницы, мы будем предполагать, что вы согласились с этими требованиями». Это один из вариантов, но на деле это способ *не достичь* соглашения. Есть также риск усложнения отношений с заинтересованными лицами, от которых вы ожидали молчаливого согласия. Попытайтесь понять, почему им не хочется ставить свою подпись, и решить саму проблему.

В такой ситуации лучше двигаться вперед, но осторожно, предполагая что у вас нет одобрения упорствующих заинтересованных лиц. Задокументируйте тот факт, что определенные заинтересованные лица не завизировали требования, в своем списке рисков вместе с возможным влиянием от пропуска или неверности некоторых требований. Работу с этими лицами нужно рассматривать как часть процедур по управлению рисками. В позитивном ключе упомяните, что вы в курсе, что они пока не одобрили требования, но проект движется вперед с этими требованиями в качестве базовых, чтобы не задерживать работу. Сообщите им, что если они хотят что-то изменить, для этого есть соответствующий процесс. В сущности, вы действуете так, как будто заинтересованное лицо согласилось с требованиями, но вы четко отслеживаете отношения.

Согласование требований в проектах гибкой разработки

В проектах гибкой разработки (agile) нет формальной операции визирования. В таких проектах требования обычно хранятся в виде пользовательских историй в резерве (backlog) продукта. Владелец продукта и команда согласовывают, какие рассказы будут реализовываться в следующей итерации, в процессе совещания по планированию. Набор историй выбирается на основе приоритетов и скорости работы (продуктивности) команды. После определения и согласования набора содержащиеся в итерации истории замораживаются. Новые запрошенные изменения планируются только на будущие итерации. В проектах гибкой разработки никто не пытается получить одобрение заинте-

ресованного лица сразу на весь набор требований. В таких проектах полный набор функциональности определяется со временем, хотя концепция и другие бизнес-требования должны быть определены с самого начала. Подробнее о работе с требованиями в проектах гибкой разработки см. главу 20.

Однажды мне пришлось работать в проекте, где клиент запросил подписание требований несмотря на то, что использовалась гибкая методика разработки. Команде пришлось проявить изобретательность, чтобы сделать это в контексте, который обычно не предусматривает визирования. Команда аналитиков тесно работала с пользователями, чтобы собрать и проверить требования в форме пользовательских историй и других моделей, таких как потоки процессов и таблицы состояний. Мы просили пользователей «завизировать», что на тот момент времени не было отсутствующих *известных им* важных требований и никаких *известных им* серьезных проблем с написанным нами. Так как пользователи участвовали в деятельности по определению требований, разработчики работали над решением, которое не могло оказаться слишком отходящим от базовых требований. Но такое понимание «визы» оставляет открытым право пользователей осознать, что им требуется добавить что-то новое или что-то исправить в существующих требованиях.

В отличие от исторически сложившегося значения визы (или подписи) — «одобрение и заморозка всех требований с первого до последнего», — этот подход никого не загоняет в угол и не заставляет чувствовать, что дальнейшая жизнь будет зависеть от объемного документа требований, который полностью так и не удалось понять. И клиентов не вынуждают признать, что требования близки к идеальным и все сделано правильно с первого раза. Такая версия визирования оставляет дух гибкой разработки. Как и в случае с описанным ранее процессом подписания, суть заключается в достижении соглашения относительно конкретного набора *базовых* требований, которые будут реализованы в следующем цикле, с общим пониманием, что такое соглашение на самом деле означает.

В проектах гибкой разработки владелец продукта публично принимает или отвергает требования на итерацию, которые представляют собой набор историй и связанных критериев принятия и приемочных тестов. Конечное «подписание» заключается в приемке работающего и протестированного ПО, полученного на данной итерации.

Как сказала консультант Нанетт Браун (Nanette Brown): «Даже в среде гибкой разработки принцип визирования может играть полезную роль. Гибкость говорит нам «полагаться на изменения», но сама концепция изменения существует только в связке с точкой отсчета. Даже в команде, где ее члены тесно общающихся друг с другом, у людей могут быть разные интерпретации текущих планов и состояния. Предложение об изменении, представленное на рассмотрение одним человеком, другой может посчитать уже согласованным. Однако если процедуру визирования считать как упрощенный церемониал признания, что «мы находимся на таком и таком этапе», то это нормально. Такое утверждение не означает, что завтра мы не можем быть

в другом месте, но как минимум это обеспечивает взаимопонимание и общую точку отсчета».

Что дальше?

- Определите, кто из клиентов, включая конечных пользователей, предоставит бизнес-требования и пользовательские требования к проекту. Какие пункты «Билля о правах» и «Билля об обязанностях» эти клиенты понимают, принимают и используют на практике, а какие — нет?
- Обсудите «Билль о правах» со своими основными клиентами и узнайте, нет ли у них ощущения бесправности. Обсудите «Билль об обязанностях» и выясните, какие из обязанностей клиенты принимают. Измените «Билль о правах» и «Билль об обязанностях» так, чтобы все стороны приняли концепцию дальнейшего сотрудничества. Отслеживайте, все ли заинтересованные лица соблюдают баланс прав и обязанностей.
- Если вы участвуете в проекте по разработке ПО в качестве клиента и чувствуете, что ваши права попираются, обсудите это с менеджером проекта или аналитиком требований. Подтвердите свое согласие выполнять «Билль об обязанностях», так как вы стремитесь наладить более тесные и дружеские рабочие отношения.
- Если в вашей организации применяется формальное подписание, подумайте, что оно должно означать в современных условиях. Совместно с руководством разработчиков и клиента (или отдела маркетинга) выработайте соглашение о том, что подпись должна означать в вашем процессе одобрения требований.
- Укажите один пример текущего или прошлого проекта, в котором отсутствовал достаточный уровень участия клиента. Проанализируйте, как это повлияло на проект. Подумайте, можете ли вы количественно оценить риск в плане поздних изменений требований, времени, потраченного на исправление продукта после поставки или утраченных бизнес-возможностей. Используйте этот опыт в будущем в качестве наглядного материала и для аргументации перед клиентом критической важности его участия.

Глава 3

Рекомендуемые приемы формулирования требований

«Добро пожаловать в команду, Сара, — сказала менеджер проекта Кристин. — Мы очень надеемся, что ты поможешь нам с требованиями в этом проекте. Я так понимаю, что на предыдущем месте ты работала бизнес-аналитиком. Есть ли у тебя идеи насчет того, с чего надо начать?»

«Ну, — ответила Сара, — Я планировала проинтервьюировать ряд пользователей и узнать, что они хотят. После этого я запишу то, что они мне сообщат. Этого будет достаточно для начала работы разработчиков. В основном мы именно так и поступали ранее. Вы не подскажете, с какими пользователями я могла бы поговорить?»

«Хм, ты действительно считаешь, что этого будет достаточно для проекта такого типа?» — спросила Кристин. — Мы пробовали такой подход ранее, но он оказался не очень хорошим. Я надеялась, что у тебя есть какие-то идеи насчет удачных приемов с предыдущих мест работы, которые окажутся лучше, чем просто интервьюирование нескольких пользователей. Если ли какие-то приемы, которые ты нашла особенно полезными?»

Сара смешалась: «Я не знаю других способов работы с требованиями кроме разговора с пользователями и попыткой записать четкие спецификации на основе сказанного ими. На своем предыдущем рабочем месте я прикладывала максимум усилий и использовала свои знания и навыки бизнеса. Я посмотрю, что можно сделать».

Каждому специалисту по ПО нужно завести набор приемов, которые он может использовать для решения задач, возникающих в процессе проекта. Практик, у которого нет такого набора, вынужден изобретать подходы, исходя из разумного выбора на той или иной стадии. Такие спонтанные методы редко дают хорошие результаты. Некоторые люди пропагандируют методологии разработки ПО — пакеты методик, которые, как ожидается, предоставляют собой целостный набор решений проблем, возникающих в процессе реализации проекта. Однако простое следование инструкции — стандартному процессу, который, как предполагается, работает в любой ситуации — тоже не всегда дает отличные результаты. Мы считаем, что более эффективно определять и применять рекомендуемые в отрасли приемы. Подход на основе ре-

комендуемых приемов позволяет создать набор различных методик, которые применимы к решению самых разных проблем.

Понятие рекомендаций довольно спорно: кто решает, что лучше, и на каком основании? Можно создать группу экспертов или исследователей и проанализировать проекты различных организаций. Таким образом экспертам удастся выявить приемы, которые обычно эффективно применялись в успешных проектах, а в неудачных — показали себя плохо или не применялись вообще. В процессе работы эксперты согласованно определяют, какие действия неизменно дают превосходный результат; обычно их называют *рекомендуемыми приемами* (best practices).

Табл. 3-1. Приемы формулирования требований

Сбор информации	Анализ	Спецификации	Проверка
• Определите концепцию продукта и границы проекта • Определите классы пользователей • Выделите из пользователей ярых сторонников продукта • Создайте фокус-группы • Определите пользовательские требования • Определите системные события и реакцию на них • Проведите интервью для выявления требований • Проведите семинары по выявлению требований • Наблюдайте за пользователями на рабочих местах • Раздайте опросные листы • Выполните анализ документов • Изучите отчеты о проблемах • Повторно задействуйте существующие требования	• Смоделируйте среду приложения • Создайте прототипы • Проанализируйте осуществимость • Расставьте приоритеты для требований • Создайте словарь данных • Смоделируйте требования • Проанализируйте интерфейсы • Распределите требования по подсистемам	• Используйте шаблон спецификации требований • Определите источники требований • Задайте каждому требованию уникальный идентификатор • Задокументируйте бизнес-правила • Определите атрибуты качества	• Изучите документы с требованиями • Протестируйте требования • Определите критерии приемлемости • Смоделируйте требования

Управление требованиями	Обучение	Управление проектом
• Определите процесс управления изменениями • Проанализируйте, какое влияние оказывают изменения • Определите базовую и контрольную версии наборов требований • Отслеживайте хронологию изменений • Отслеживайте состояние требований • Отлеживайте проблемы с требованиями • Создайте матрицу связей требований • Используйте средство управления требованиями	• Обучите аналитиков требований • Ознакомьте представителей пользователей и менеджеров с требованиями • Обучите разработчиков основам предметной области • Определите процесс разработки требований • Создайте словарь терминов	• Выберите соответствующий цикл разработки проекта • Планируйте подход к работе с требованиями • Оцените объем работ по реализации требований • Планируйте на основании требований • Определите лиц, ответственных за принятие решений по требованиям • Своевременно пересматривайте обязательства • Управляйте рисками, касающимися требований • Отслеживайте объем работ по реализации требований • Делайте выводы из полученного опыта

В табл. 3-1 перечислено приблизительно 50 приемов; они разделены на 7 категорий. Некоторые приемы относятся к нескольким категориям, но в таблице они указаны только один раз. Большинство этих приемов способствуют более эффективной коммуникации между заинтересованными лицами проекта. Заметьте: глава называется «Рекомендуемые приемы формулирования требований», а не «Лучшие приемы». Сомневаюсь, что когда-либо будет проведена систематическая оценка всех их на предмет выбора лучшего. Тем не менее, многие практики, и я в том числе, считают данные приемы эффективными (Sommerville и Sawyer, 1997; Hofmann и Lehner, 2001; Gottesdiener, 2005; IIBA, 2009).

В этой главе кратко описывается каждый прием, а также указываются ссылки на другие главы и источники. Приемы годятся не для всех ситуаций, и поэтому стоит не слепо следовать сценарию, а руководствоваться трезвым расчетом, здравым смыслом и опытом. Даже самые лучшие приемы должны вдумчиво отбираться, применяться и внедряться в соответствующих ситуациях опытным аналитиком. Различные приемы могут подходить для понимания требований в разных частях проекта. Например, варианты использования системы и прототипы пользовательских интерфейсов могут пригодиться на стороне клиента, а на стороне сервера важнее анализ интерфейсов.

Люди, которые применяют или играют ведущую роль в применении этих приемов, разные в разных приемах и проектах. Аналитик будет играть ключевую роль во многих из них, но не в каждом проекте есть аналитик. Заказчик продукта

может применить один из приемов в проекте гибкой разработки (agile). Но другие приемы относятся к сфере ответственности менеджера проекта. Подумайте, кто в вашей текущей команде будет играть ведущую роль или участвовать в применении приемов, которые вы выбрали для следующего проекта.

> **Внимание!** Ни один из этих приемов не будет работать, если вы имеете дело с неблагоразумными людьми. Клиенты, менеджеры и специалисты по ИТ иногда бывают недоговороспособными, но может оказаться, что они просто недостаточно информированы. Они могут не знать, почему вы хотите применить те или иные приемы, или чувствовать себя неуютно с незнакомыми терминами и действиями. Попробуйте просветить своих коллег насчет своих приемов и почему вы хотите применить именно их, а также почему для их же собственного блага стоит сотрудничать с остальными членами команды.

Каркас процесса создания требований

Как вы знаете из главы 1, разработка требований состоит из выявления, анализа, документирования и проверки. Не ждите, что все действия удастся выполнить последовательно и за один проход. На практике эти действия выполняются попеременно, поэтапно и повторяются (рис. 3-1). «Поступательная очистка деталей» — вот, что должно быть основным девизом при разработке требований и переходе от начальных концепций к более точному пониманию и выражению.

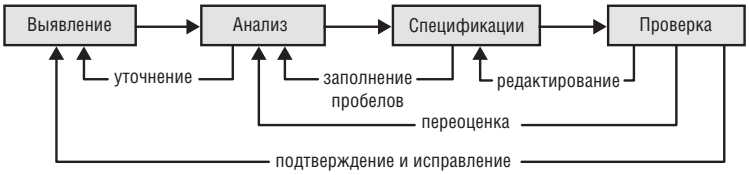


Рис. 3-1. Итеративный процесс формулирования требований

Будучи аналитиком, вы будете задавать клиентам вопросы, слушать их ответы и наблюдать, что они делают (этап выявления требований). Вы обработаете эту информацию и разберетесь в ней, классифицируете ее на разные категории и свяжете потребности клиента с возможными программными требованиями (этап анализа). В результате анализа может оказаться, что нужно уточнить некоторые требования, поэтому вы возвращаетесь назад и дополняете набор требований. После этого вы структурируете информацию от пользователей и выведенные требования в виде письменных требований-утверждений и диаграмм (спецификация). При письменной фиксации может потребоваться вернуться назад и выполнить дополнительный анализ, чтобы закрыть пробелы в знаниях. Затем вы просите ряд заинтересованных лиц подтвердить, что представленное вами точно и полно отражает потребности, и исправить ошибки (проверка). Это выполняется по отношению к набору требований, которые важнее и своевременнее всего для начала разработки ПО. В процессе проверки может потребоваться переписать некото-

рые неясные требования, повторить некоторые действия анализа или даже вернуться и выполнить дополнительное выявление требований. После этого вы переходите к следующему этапу проекта и весь цикл повторяется. Такой итеративный процесс продолжается на всем протяжении разработки требований и возможно — как в проектах гибкой разработки — на протяжении всего времени проекта.

Из-за разнообразия проектов по разработке ПО и организационных культур единого, шаблонного подхода к созданию требований не существует. На рис. 3-2 показан каркас создания требований, который с разумными исправлениями подойдет для большинства проектов. Предшественником процесса показанного на рис. 3-2 является бизнес-потребность или рыночная возможность. Как правило, действия выполняются в основном по порядку, однако сам процесс не является строго последовательным. Первые семь действий обычно однократно выполняются на ранних стадиях работы над проектом (тем не менее, команде разработчиков придется периодически проверять эти действия). Остальные необходимы для каждого очередного выпуска или этапа работы над проектом. Многие из этих действий могут выполняться итеративно и попеременно. Например, шаги 8, 9 и 10 можно выполнять небольшими порциями, выполняя пересмотр (шаг 12) после каждой итерации.

Рис. 3-2. Примерный процесс создания требований

Пятый подраздел разработки требований — управление требованиями. К управлению требованиями относятся приемы работы с уже готовыми требованиями. Эти приемы включают управление версиями и определение ба-

зовых требований, управление изменениями, отслеживание состояния требований и отслеживание связей требований с другими элементами системы. Управление требованиями — это деятельность низкой интенсивности, происходящая на всем протяжении проекта.

На рис. 3-3 показано, как распределяются работы по созданию требований по этапам разработки продукта в жизненном цикле ПО общего вида. Общий объем работ по работе с требованиям может быть практически таким же, как и в проектах сравнимого размера с другими жизненными циклами, но распределение работ по времени может сильно отличаться. В чистом водопадном цикле планируется только один основной выпуск, поэтому основной объем работы с требованиями приходится на начало проекта (сплошная линия на рис. 3-3). Такой подход используется в очень многих проектах, но подходит для немногих. Но даже если вы спланируете традиционный этап «выявления требований» в начале проекта, после чего требования используются для проектирования, нужно рассчитывать на выполнение небольшого объема работ с требованиями на протяжении всего проекта.

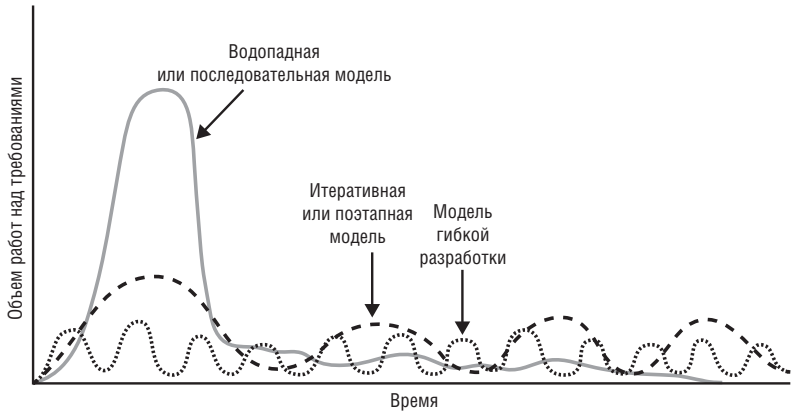


Рис. 3-3. Распределение работ с требованиями на протяжении жизненного цикла проекта в разных моделях разработки

В проектах, в которых используется итеративный процесс разработки, такой как Rational Unified Process (Jacobson, Booch и Rumbaugh, 1999), работа с требованиями ведется в каждой итерации процесса разработки, причем на первую итерацию приходится б**о**льший объем работы (пунктирная линия на рис. 3-3). Это же происходит, если вы запланировали серию поэтапных выпусков, в каждом из которых предоставляется значительная часть конечной функциональности продукта.

В проектах гибкой разработки (agile) планируется выпускать функциональность каждые несколько недель (Larman, 2004). В таких проектах работа над требованиями выполняется часто, но небольшими объемам, как показано на рис. 3-3 точечной линией. Работа начинается с выявления пользовательских требований в форме пользовательских историй, которые описывают основные задачи, которые пользователи хотят решить с помощью систе-

мы. В этом подходе из историй нужно извлечь достаточно информации, чтобы можно было оценить объем и определить приоритеты разработки. Приоритизация пользовательских требований позволяет определить, какие из них назначить на те или иные этапы разработки, которые называются итерациями или спринтами. Более подробно выделенные на ту или иную итерацию требования изучаются, когда разработка подойдет к этой итерации.

Независимо от используемой в проекте модели, надо при каждом выпуске или итерации задавать себе вопрос, какие из действий, описанных на рис. 3-2, позволят увеличить ценность и снизить риск. По завершении шага 17 для любой части требований можно приступать к построению соответствующей части системы. Повторите шаги с 8 по 17 со следующим набором требований, который ляжет в основу следующего выпуска или итерации.

Выявление требований

В главе 1 обсуждались три уровня требований: бизнес-уровень, пользовательский уровень и функциональный. Они собираются из разных источников на различных этапах работы над проектом, имеют различные цели и аудиторию и должны документироваться по-разному. Также следует выявить и собрать нефункциональные характеристики, например предполагаемое качество в разных измерениях, из соответствующих источников. Далее приводятся приемы, которые помогут собрать информацию требований самых разных типов.

Определение концепции продукта и границ проекта Документ о концепции и границах содержит бизнес-требования к продукту. Описание концепции позволяет всем заинтересованным лицам в общих чертах понять назначение продукта. Границы проекта определяют, что следует реализовать в этой версии, а что — в следующих. Концепция и границы — хорошая база для оценки предлагаемых требований. Концепция продукта должна оставаться от версии к версии относительно стабильной, но для каждого выпуска необходимо составить отдельный документ о границах. Подробнее см. главу 5.

Определение классов пользователей и их характеристик Чтобы не упустить из виду потребности отдельных пользователей, выделите их в группы. Например, по частоте работы с ПО, используемым функциям, уровню привилегий и опыту работы. Опишите их обязанности, местоположение и личные характеристики, способные повлиять на архитектуру продукта. Создайте архетипы пользователей — описания выдуманных людей, которые будут представлять конкретные классы пользователей. Подробнее см. главу 6.

Выбор сторонника продукта (product champion) в каждом классе пользователей Это человек, который сможет точно передавать настроения и нужды клиентов. Он представляет потребности определенного класса пользователей и принимает решения от их лица. В случае разработки внутрикорпоративных информационных систем, когда все пользователи — ваши коллеги, такого че-

ловека выбрать проще. При коммерческой разработке порасспросите клиентов или используйте площадки бета-тестирования. Подробнее см. главу 6.

Проведение фокус-групп типичных пользователей Создайте группы типичных пользователей предыдущих версий вашего продукта или похожих. Выясните у них подробности функциональности и качественных характеристик разрабатываемого продукта. Фокус-группы особенно ценны при разработке коммерческих продуктов, когда приходится иметь дело с большой и разнородной клиентской базой. В отличие от сторонников продукта, у фокус-групп обычно нет полномочий на принятие решений. Подробнее см. главу 7.

Работа с пользователями для выяснения назначения продукта Выясните у представителей пользователей, какие задачи им требуется выполнять средствами ПО и какую пользу они хотят из него извлечь. Пользовательские требования могут выражаться в форме сценариев использования, задач или пользовательских историй. Обсудите то, каким образом клиент должен взаимодействовать с системой для выполнения каждой такой задачи. Подробнее см. главу 8.

Определение системных событий и реакции на них Перечислите возможные внешние события и ожидаемую реакцию системы на них. Существует три класса внешних событий. Это могут быть сигналы и данные, получаемые от внешнего оборудования. Есть также временны*е* или основанные на времени события, вызывающие ответную реакцию, например ежевечерняя передача данных во внешний канал данных, инициируемая системой ежевечернее в одно и то же время. В бизнес-приложениях бизнес-события напрямую инициируют выполнение бизнес-задач. Подробнее см. главу 12.

Проведение интервью для выявления требований Интервью можно проводить в формате «один на один» или с небольшой группой заинтересованных лиц. Это эффективный способ выявления требований без слишком большого отвлечения заинтересованного лица от его работы, потому на таких встречах обсуждаются только конкретные требования, имеющие значение для интервьюируемого заинтересованного лица. Интервью полезны для раздельного выявления требований у людей при подготовке к семинарам, где они встречаются для разрешения возможных конфликтов. Подробнее см. главу 7.

Проведение совместных семинаров Совместные семинары по выявлению требований, где тесно сотрудничают аналитики и клиенты — отличный способ выявить нужды пользователей и составить наброски документов с требованиями (Gottesdiener, 2002). Такие семинары иногда называют JAD-семинарами (Joint Application Design) (Wood и Silver, 1995). Подробнее см. главу 7.

Наблюдение за пользователями на рабочих местах Наблюдая за работой пользователей, выявляют контекст возможного применения нового продукта. Простые диаграммы рабочих потоков, а также диаграммы потоков данных позволяют выяснить этапы и принимаемые решения, а также то, как взаимодействуют различные группы пользователей. Документируя ход бизнес-процесса, удается определить требования к системе, рассчитанной на поддержку этого процесса. Подробнее см. главу 7.

Раздача опросных листов Это один из способов определения потребностей больших групп пользователей. Опросные листы удобны при работе с любыми большими группами пользователей, но особенно полезны в распределенных группах. Качественные вопросы позволяют быстро выявить аналитическую информацию о потребностях. На основе результатов опросных листов можно более целенаправленно применять дополнительные усилия. Подробнее см. главу 7.

Анализ документов Имеющаяся документация может помочь выявить, как системы работают сейчас или что они должны делать. К документации относится вся письменная информация о текущих системах и бизнес-процессах, спецификации требований, исследования конкурентов и руководства имеющихся коммерческих программных пакетов. Просмотр и анализ этих документов может помочь выявить функциональность, которая должна остаться и которая больше не нужна, а также определить, как сейчас люди выполняют свою работу, что предлагают конкуренты и что говорят поставщики о том, что должно делать их ПО. Подробнее см. главу 7.

Изучение отчетов о проблемах работающих систем с целью поиска новых идей Поступающие от клиентов отчеты о проблемах и предложения о расширении функциональности — отличный источник идей о возможностях, которые можно реализовать в следующей версии или новом продукте. За подобной информацией стоит обратиться к персоналу службы поддержки.

Повторное использование требований Если необходимая клиенту функциональность аналогична уже реализованной в другом продукте, подумайте, готовы ли клиенты гибко пересмотреть свои требования для использования существующих компонентов. Требования, соответствующие бизнес-правилам компании, можно применить в нескольких проектах, например требования к безопасности, определяющие порядок доступа к приложениям, и требования, соответствующие требованиям государственных органов, например Закону о гражданах США с ограниченными возможностями (Americans with Disabilities Act). К другим кандидатам на повторное использование относятся глоссарии, модели и определения данных, профили заинтересованных лиц, описаний классов и архетипы пользователей. Подробнее см. главу 18.

Анализ требований

Анализ требований подразумевает их очистку, гарантирующую, что требования понимают все заинтересованные лица, а также тщательное исследование требований на предмет ошибок, пробелов и других недостатков. Кроме того, анализ включает разбиение высокоуровневых требований на более детальные, создание прототипов, анализ осуществимости и согласование приоритетов. Цель анализа — достаточно качественно и подробно описать требования, позволяющие менеджерам реалистично оценить все затраты на проект, а техническому персоналу — начать проектирование, разработку и тестирование.

Очень ценно представлять отдельные требования несколькими способами, например в текстовой и графической форме или одновременно в виде требований и тестов (Wiegers, 2006). Это позволит выявить их особенности и проблемы, не заметные при представлении одним способом. Также это помогает всем заинтересованным лицам выработать согласованное представление об итогах разработки продукта.

Моделирование среды приложения Контекстная диаграмма — простая модель анализа, отображающая место новой системы в соответствующей среде. Она определяет границы и интерфейсы между разрабатываемой системой и сущностями, внешними для этой системы, например пользователями, устройствами и прочими информационными системами. Карта экосистемы показывает различные системы в пространстве решения, которые взаимодействуют друг с другом, а также природу их взаимосвязей (Beatty и Chen, 2012). Подробнее см. главу 5.

Создание пользовательского интерфейса и технических прототипов Если разработчики или пользователи не совсем уверены насчет требований, создайте прототип — частичную, возможную или предварительную версию продукта, которая сделает концепции и возможности более осязаемыми. Оценка прототипа поможет разработчикам и пользователям достичь взаимопонимания по решаемой проблеме, а также проверить требования. Подробнее см. главу 15.

Анализ осуществимости требований Аналитик совместно с разработчиками должен определить, насколько возможно реализовать каждое требование при разумных затратах и с приемлемой производительностью в предполагаемой среде. Это позволяет заинтересованным лицам понять риски, связанные с реализацией каждого требования, включая конфликты с другими требованиями, зависимость от внешних факторов и препятствия технического характера. Технически нереализуемые или очень дорогие в реализации требования можно попытаться упростить и в таком виде включить в проект для достижения бизнес-целей.

Определение приоритетов требований Очень важно определить приоритеты требований, чтобы быть уверенным, что команда реализует самую важную и своевременную функциональность в первую очередь. Воспользуйтесь аналитическим подходом и определите относительные приоритеты реализации функций продукта, решаемых задач, пользовательских историй или отдельных требований. На основании приоритетов установите, в какой версии будет реализована та или иная функция или набор требований. В ходе работы над проектом периодически корректируйте приоритеты в соответствии с потребностями клиента, условиями рынка и бизнес-целями. Подробнее см. главу 16.

Создание словаря данных В нем соберите определения всех элементов и структур данных, связанных с системой, что позволит всем участникам проекта использовать согласованные определения данных. На стадии работы над требованиями словарь должен содержать определения элементов данных, относящихся к предметной области, чтобы клиентам и разработчикам было проще общаться. Подробнее см. главу 13.

Моделирование требований Модель анализа представляет собой диаграмму, которая визуально отображает требования, в отличие от текстового представления списка функциональных требований. Модели позволяют выявить некорректные, несогласованные, отсутствующие и избыточные требования. К таким моделям относятся диаграммы потоков данных, диаграммы «сущность-связь», диаграммы перехода состояний, таблицы состояний, карты диалоговых окон, деревья решений и другие (Beatty и Chen, 2012). Подробнее о моделировании см. главы 5, 12 и 13.

Анализ интерфейсов между системой и внешним миром Во всех программных системах существуют связи с другими частями, реализованные как внешние интерфейсы. Информационные системы имеют пользовательские интерфейсы и часто обмениваются данными с другими программными системами. Встроенные системы обеспечивают связь между программными и аппаратными компонентами. В приложениях с подключением к сети есть коммуникационные интерфейсы. Анализ всего этого позволит обеспечить гладкую интеграцию вашего приложения в среду. Подробнее см. главу 10.

Распределение требований по подсистемам Требования к сложному продукту, включающему несколько подсистем, следует соразмерно распределять между программными, аппаратными и операторскими подсистемами и компонентами. В качестве примера такого продукта можно привести систему доступа для защиты здания, которая включает магнитные или оптические бейджи, сканеры, видеокамеры и регистраторы, дверные замки и охранников. Подробнее см. главу 26.

Спецификации требований

Суть спецификации требований заключается в документировании требований различных типов единообразным, доступным и поддающимся проверке способом так, чтобы они были понятными целевой аудитории. Зафиксировать бизнес-требования можно в положении о концепции и границах. Пользовательские требования обычно представляют в виде вариантов применения или пользовательских историй. Подробные функциональные и нефункциональные требования к ПО фиксируются в спецификации требований к ПО (SRS) или другом хранилище, таком как средство управления требованиями.

Внедрение шаблонов документов требований Создайте стандартные шаблоны для документирования требований к ПО в вашей организации, такие как документ концепции и границ в главе 5, шаблон варианта использования в главе 8 и шаблон SRS в главе 10. Шаблон предоставляет согласованную структуру, позволяющую фиксировать описания разной информации, касающейся требований. Даже если вы не храните требования в традиционной документарной форме, шаблон напомнит вам о разных видах информации требований, которые нужно собрать и записать.

Определение источников требований Чтобы гарантировать, что все заинтересованные лица понимают, зачем нужно то или иное требование, выявите источники всех требований. Это может быть вариант использования или другая информация от пользователей, системное требование высокого уровня или бизнес-правило. Указав всех лиц, заинтересованных в каждом требовании, вы будете знать, к кому обратиться при поступлении запроса на изменение. Источники требований устанавливают на основе связей или определяют для этой цели атрибут требования. Подробнее об атрибутах требований см. главу 27.

Присвоение уникальных идентификаторов всем требованиям Выработайте соглашение о присвоении уникальных идентификаторов требованиям. Соглашение должно обеспечивать возможность дополнения, удаления и изменения требований. Присвоение идентификаторов позволяет отслеживать требования и фиксировать вносимые изменения. Подробнее см. главу 10.

Документирование бизнес-правил К бизнес-правилам относятся корпоративные политики, правительственные распоряжения и алгоритмы вычислений. Ведите список бизнес-правил отдельно от требований проекта, поскольку правила обычно существуют вне рамок конкретного проекта. То есть бизнес-правила нужно рассматривать как актив уровня предприятия, а не проекта. Для выполнения некоторых приходится создавать реализующие их функциональные требования, и поэтому необходимо определить связь между этими требованиями и соответствующими правилами. Подробнее см. главу 9.

Определение нефункциональных требований Можно реализовать решение, которое делает ровно то, что ожидается, но не реализует ожиданий пользователей в плане качестве. Чтобы избежать подобной проблемы, нужно выйти за рамки обсуждения функциональности, чтобы понять характеристики качества, которые важны для успеха проекта. К этим характеристикам относятся производительность, надежность, удобство использования, возможность внесения изменений и многое другое. Информация от клиента об относительной важности эти атрибутов качества позволяет разработчику принимать соответствующие проектные решения. Также нужно указать требования внешних интерфейсов, ограничения дизайна и реализации, проблемы интернационализации и другие нефункциональные требования. Подробнее см. главу 14.

Проверка требований

Проверка гарантирует, что все положения требований корректны, отражают желаемые качественные характеристики и удовлетворяют потребностям клиента. Может оказаться, что требования, в спецификации требований к ПО выглядевшие превосходно, при реализации оказываются содержащими двусмысленности и неопределенности. Если требования должны стать надежной основой для проектирования и итоговой проверки системы посред-

ством системного тестирования или приемочного тестирования пользователями, эти проблемы необходимо устранить. Подробнее см. главу 17.

Рецензирование требований Рецензирование и, в частности, более жесткая процедура, называемая *освидетельствование* (inspection), — один из наиболее ценных приемов обеспечения качества ПО (Wiegers, 2002). Соберите небольшую команду рецензентов, представляющих различные области, (таких как аналитик, клиент, разработчик и тестировщик), и внимательно изучите письменные требования, аналитические модели и связанную информацию на предмет ошибок. Также ценно неформальное предварительное рецензирование во время разработки требований. Важно обучить членов команды, как эффективно выполнять рецензирование требований, а также внедрить в организации процесс рецензирования. Подробнее см. главу 17.

Тестирование требований Тесты составляют альтернативное представление требований. При написании тестов приходится задумываться, как определить, что ожидаемая функциональность реализована правильно. Создавайте тесты на основе пользовательских требований для документирования ожидаемого поведения продукта в определенных условиях. Пройдите тесты с клиентами, чтобы убедиться, что они отражают ожидания пользователей. Свяжите тесты с функциональными требованиями, чтобы убедиться, что ни одно требование не было упущено и что каждому соответствует тест. Используйте тесты для проверки правильности моделей анализа и прототипов. В проектах гибкой разработки часто создают приемочные тесты вместо подробных функциональных требований. Подробнее см. главу 17.

Определение критериев приемки Предложите пользователям описать, как они собираются определять соответствие продукта своим потребностям и его пригодность к работе. Критерии приемки сочетают в себе прохождение продуктом приемочных тестов, основанных на требованиях, удовлетворение определенным нефункциональным требованиям, отслеживание открытых дефектов и проблем, наличие инфраструктуры и обучения, готовых для успешного развертывания и многое другое. Подробнее см. главу 17.

Моделирование требований Доступны платные средства, позволяющие команде проекта смоделировать предлагаемую систему на месте или в дополнение к задокументированным в письменной форме спецификациям. Моделирование это следующий уровень после прототипов, который позволяет аналитику в процессе взаимодействия с пользователями быстро построить макет системы. Пользователи могут работать с моделью системы, выбирая дизайн и проверяя требования до того, как они застынут в коде. Моделирование не заменяет вдумчивого выявления и анализа требований, но является мощным вспомогательным средством.

Управление требованиями

Начальные требования неизбежно корректируются в процессе работы клиентами, менеджерами, специалистами по маркетингу, разработчиками и

другими лицами. Для эффективного управления требованиями необходим процесс, позволяющий предлагать изменения и оценивать их возможную стоимость и влияние на проект, а также гарантировать, что соответствующие заинтересованные лица принимают разумные бизнес-решения о том, какие из предложенных изменений внедрить в требования.

Для эффективного управления требованиями жизненно необходимы выверенные способы управления конфигурацией. Для управления требованиями годятся те же утилиты для управления версиями, которые вы используете для управления базой кода. А лучше всего хранить требования в средстве управления требованиям, которое специально предназначено для этого.

Определение процесса управления изменениями Вместо того, чтобы подавлять изменения или надеяться, что они не появятся, примите как данное тот факт, что они будут и создайте механизм управления ими, чтобы вал изменений не приводил к хаосу. Процесс изменений должен определять порядок представления, анализа и утверждения или отклонения изменений требований. Применяйте его для управления всеми предлагаемыми изменениями. В контексте процесса управления изменениями полезно использовать коммерческие средства отслеживания дефектов. Из представителей заинтересованных в проекте лиц организуйте совет по управлению изменениями, который будет получать информацию о предполагаемых изменениях требований, оценивать ее, решать, какие изменения принять, а какие отклонить, и определять, в какой версии продукта будет внедрена та или иная функция. Подробнее см. главу 28.

Анализ влияния изменений требований Анализ влияния изменений — важный элемент процесса внесения изменений, который помогает совету по управлению изменениями принимать обоснованные решения. Оцените, как каждое предлагаемое изменение требований повлияет на проект. На основе матрицы связей выявите другие требования, элементы архитектуры, исходный код и сценарии тестирования, которые, возможно, придется изменить. Определите, что необходимо для реализации изменений, и оцените затраты на реализацию. Подробнее см. главу 28.

Создание базовой версии и управление версиями требований Базовая версия содержит требования, утвержденные для реализации в конкретной версии или итерации продукта. После определения базовых требований изменения можно вносить только в соответствии с процессом управления изменениями. Присвойте всем версиям спецификации требований уникальные идентификаторы, чтобы избежать путаницы между черновыми вариантами и базовыми версиями, а также между предыдущей и текущей версиями требований. Подробнее см. главы 2 и 27.

Ведение журнала изменений требований Ведите хронологию изменений требований. Иногда нужно вернуться к предыдущей версии требования или узнать, как требование пришло в текущее состояние. Фиксируйте даты изменения требований, сами коррективы, их причины, а также лиц, вносивших изменения. Автоматизировать эти задачи позволяет утилита управления версиями или коммерческая утилита управления требованиями.

Отслеживание состояния всех требований Создайте БД, включающую по одной записи для каждого дискретного функционального требования. Занесите в БД ключевые атрибуты каждого требования, включая его состояние (например «предложено», «одобрено», «реализовано» или «утверждено»), чтобы в любой момент вы могли узнать количество требований в каждом состоянии. Отслеживание состояния каждого требования по мере его движения через разработку и системное тестирование предоставляет информацию о состоянии проекта в целом. Подробнее см. главу 27.

Отслеживание проблем с требованиями Когда занятые люди работают над сложным проектом, легко упустить многие возникающие проблемы, в том числе вопросы требований, нуждающиеся в разрешении, незаполненные пробелы и проблемы, возникающие при рецензировании требований. Средства отслеживания дефектов могут обеспечить, чтобы эти вещи не остались без внимания. Назначьте по одному ответственному на каждую проблему. Отслеживайте состояние проблем с требованиями, чтобы быть в курсе общего состояния дел с требованиями. Подробнее см. главу 27.

Создание матрицы связей требований Всегда полезно, а иногда и обязательно, собирать наборы связей, которые соединяют отдельные функциональные требования с элементами архитектуры и кода, которые реализуют эти требования, а также с тестами для их проверки. Такая *матрица связей требований* помогает при проверке, все ли требования реализованы и проверены. Она также полезна на этапе обслуживания, когда требование нужно изменить. Матрица связей требований позволяет также сопоставить функциональные требования с требованиями более высоких уровней, на основе которых они созданы, и с другими родственными требованиями. Заполняйте эту таблицу в ходе, а не в конце работы над проектом. Обязательно использовать для этих целей специальные программные средства, а исключение можно сделать только для самых мелких проектов. Подробнее см. главу 29.

Использование средств управления требованиями Коммерческие утилиты управления требованиями позволяют хранить различные типы требований в базе данных. Эти средства помогают автоматизировать многие задачи по управлению требованиями, описанные ниже. Подробнее см. главу 30.

Обучение

Обязанности аналитика требований в проекте могут выполнять разные члены команды, но очень немногие из них формально обучались разработке требований. Бизнес-анализ — специализированная и непростая роль, для которой требуются особые знания (IIBA, 2009). Как и в других технических дисциплинах, опыт незаменим и бесценен. Нельзя ожидать, что люди от природы обладают навыками активного общения, которые нужны при разработке требований. Обучение позволяет повысить профессиональные навыки сотрудников, выполняющих роли аналитиков, но не может компенсировать нехватку навыков межличностного общения и отсутствие интереса к делу.

Обучение аналитиков требований Всем членам команды, которые будут исполнять функции аналитиков, необходимо научиться приемам формулирования требований — это обычно занимает несколько дней. Это позволит им получить надежную базу, на основе которой они смогут набираться собственного опыта и повышать квалификацию на продвинутых курсах. Квалифицированный аналитик требований терпелив и методичен, обладает навыками межличностного общения и коммуникативными навыками, сведущ в предметной области и знает множество способов формулирования требований к ПО. Подробнее об этой важной роли см. главу 4.

Ознакомление заинтересованных лиц с требованиями Наиболее эффективны занятия, на которых присутствуют заинтересованные лица, представляющие много функциональных областей проекта, а не только аналитики. Пользователи, которые будут принимать участие в разработке ПО, должны пройти непродолжительный тренинг (один-два дня), чтоб научиться формулировать требования. Он полезен и для менеджеров по разработке и по работе с клиентами. Коллективное участие различных заинтересованных лиц в обучении работе с требованиями к ПО может оказаться эффективным средством объединения команды. Все участники лучше поймут задачи, стоящие перед коллегами, а также что нужно членам команды друг от друга, чтобы команда в целом могла добиться успеха. Посетив мои семинары по требованиям, некоторые пользователи замечали, что стали теплее относиться к разработчикам ПО.

Ознакомление разработчиков с предметной областью Чтобы помочь разработчикам в общих чертах понять предметную область, проведите семинар, на котором познакомьте их с бизнесом клиента, терминологией и назначением создаваемого продукта. Это уменьшит вероятность путаницы, непонимания и доработок. Совершенно нелишним будет провести демонстрацию «день из жизни пользователя», на которой разработчики могут сопровождать пользователей, чтобы увидеть, как они выполняют свою работу. Можно также на время проекта назначить каждому разработчику «личного пользователя», который будет разъяснять профессиональные термины и бизнес-концепции. Лучше, если это будет горячий сторонник продукта. Сторонник продукта должен играть эту роль, как описано в главе 6.

Определение процесса разработки требований Задокументируйте последовательность шагов в ходе выявления, анализа, спецификации, проверки и управления требованиями. Наличие руководства по выполнении основных шагов позволяет аналитикам эффективно выполнять свою работу. Это также упрощает планирование задач по разработке и управлению требованиями в проекте, определению графика и необходимых ресурсов. Менеджер должен включить действия по работе с требованиями в план проекта в виде отдельных задач. Подробнее см. главу 31.

Создание бизнес-словаря Словарь со специализированными терминами из предметной области снизит вероятность непонимания. Включите в него синонимы, акронимы и сокращения, термины, имеющие несколько значений,

и термины, имеющие в предметной области и повседневной жизни разные значения. Словарь может представлять собой повторно используемый актив уровня предприятия. Разработку словаря можно поручить новым членам команды, потому что именно они больше всего будут путаться в незнакомой терминологии. Подробнее о словаре терминов см. главу 10.

Управление проектом

Способы управления проектом разработки ПО тесно связаны с работой над требованиями к нему. Менеджер проекта планирует ресурсы, графики и обязательства по проекту на основании требований, которые планируется реализовать. Альтернативный подход заключается в разбиении разработки на циклы, чтобы объем и границы работы определялись в применении к итерации фиксированной длины. Такой подход применяется в проектах гибкой разработки (agile). В рамках графика границы проекта могут обсуждаться. Это приводит к превращению «расползания границ» в «выбор границ» — заказчик продукта может потребовать что угодно и сколько угодно, но он должен определить приоритеты и по истечении времени, отведенного на итерацию, команда просто прекращает разработку. После этого планируется следующий выпуск, в котором реализуются оставшиеся требования.

Выбор цикла, или модели, разработки ПО Вашей компании следует определить несколько жизненных циклов разработки, или моделей, для проектов различного типа и различных степеней неопределенности требований (Boehm и Turner, 2004). Каждый менеджер проекта должен выбрать и использовать цикл, оптимальным образом подходящий для его проекта. Включите в цикл операции по созданию требований. По возможности определяйте и реализуйте наборы функций поэтапно, чтобы можно было периодически выпускать промежуточные версии продукта и как можно раньше предоставлять клиенту работоспособные образцы приложения (Larman, 2004; Schwaber, 2004; Leffingwell, 2011).

Планирование подхода к работе с требованиями Каждой команде проекта нужно спланировать, как вести деятельность по разработке и управлению требованиями. План выявления требований помогает обеспечить, что определение и получение информации будет происходить от надлежащих заинтересованных лиц на правильных стадиях проекта и с использованием надлежащие приемов. Аналитик и менеджер проекта должны совместно работать над тем, чтобы задачи и результаты разработки требований присутствовали в плане управления проектом. Подробнее см. главу 7.

Оценка усилий на работу с требованиями Заинтересованные лица часто интересуются тем, сколько времени займет разработка требований в проекте и какая доля всех усилий по проекту будет посвящена разработке и управлению требованиями. Естественно, это зависит от многих факторов. Учтите обстоятельства, которые могут указывать на то, что придется потратить больше или меньше времени, чем в среднем, чтобы качество требований было доста-

точным для обеспечения надежного фундамента для разработки (Wiegers, 2006). Подробнее см. главу 19.

Планы реализации проекта должны быть основаны на требованиях Разрабатывайте планы и графики работы над проектом постепенно, по мере прояснения границ и подробных требований. Начните с оценки затрат, необходимых на реализацию функциональных требований, определенных на основе первоначальных концепции продукта и границах проекта. Первые графики и оценка затрат, построенные на основе нечетких требований, окажутся крайне неточными, однако по мере детализации требований их следует уточнить. В проектах гибкой разработки поэтапная природа итераций подразумевает, что планирование включает корректировку границ проекта, чтобы они соответствовали ограничениям времени и ресурсов. Подробнее см. главы 19 и 20.

Определение лиц, ответственных за принятие решений по требованиям При разработке ПО требуется принимать много решений. Нужно устранить противоречия в полученной от пользователей информации, выбрать компоненты платного пакета, оценить запросы на изменения и т. п. Так как очень большой объем решений связан с работой с требованиями, команде проекта исключительно важно определить и наделить соответствующими полномочиями ответственных за принятие решений, желательно до того, как придется принимать свое первое значительное решение. Подробнее см. главу 2.

Пересмотр обязательств по проекту при изменении требований Команда проекта принимает обязательство предоставить определенные наборы требований в рамках определенного бюджета и за определенное время. Добавляя в проект новые требования, оцените, удается ли выполнить текущие обязательства с имеющимися ресурсами. Если нет, обсудите реалии проекта с менеджерами и согласуйте новые, более реалистичные обязательства (Wiegers, 2007; Fisher, Ury и Patton, 2011). Может также потребоваться повторно согласовывать обязательства по мере того, как требования будут развиваться от состояния неопределенности при начальной оценке до ясных и четких проверенных требований.

Анализ, документирование и управление рисками, связанными с требованиями Неожиданные события и условия могут навести хаос в проекте, если заранее не подготовиться. Выявите и задокументируйте риски, связанные с требованиями, в рамках действий по управлению рисками в проекте. Проведите мозговой штурм на предмет действий по предотвращению последствий реализации рисков и отслеживайте их изменения и эффективность. Подробнее см. главу 32.

Контроль объема работ по созданию требований Для совершенствования навыков оценки ресурсов, необходимых для работы с требованиями в будущих проектах фиксируйте усилия, прилагаемые вашей командой на разработку требований и управление проектом (Wiegers, 2006). Отслеживайте, как ваши действия по регламентации требований влияют на проект в целом, чтобы можно было оценить отдачу от этой работы. Подробнее см. главу 27.

Извлечение уроков из полученного опыта Для этого в организации следует провести ретроспективу проектов для извлечения уроков из завершенных про-

ектов или итераций текущего проекта (Kerth, 2001; Derby и Larsen, 2006; Wiegers, 2007). Ознакомление с опытом в области проблем и способов создания требований, накопленным в ходе работы над предыдущими проектами, помогает менеджерам и аналитикам требований более эффективно работать в будущем.

Начинаем применять новые приемы

В табл. 3-2 описанные выше приемы создания требований сгруппированы по их относительному влиянию на большинство проектов, а также по относительной сложности реализации. Эта классификации не абсолют — ваша ситуация может отличаться. И хотя полезны все способы, начать можно с простейших, наиболее влияющих на успех проекта и относительно простых в реализации.

Табл. 3-2. Реализация приемов формулирования требований

Влияние	**Сложность**		
	Высокая	**Средняя**	**Низкая**
Сильное	• Определите процесс разработки требований • Планируйте на основании требований • Своевременно пересматривайте обязательства	• Обучите аналитиков требований • Планируйте подход к работе с требованиями • Выделите из пользователей ярых сторонников продукта • Определите пользовательские требования • Проведите интервью для выявления требований • Определите нефункциональные атрибуты • Расставьте приоритеты для требований • Определите концепцию продукта и границы проекта • Определите процесс управления изменениями • Изучите документы с требованиями • Распределите требования по подсистемам • Используйте средство управления требованиями • Задокументируйте бизнес-правила	• Обучите разработчиков основам предметной области • Внедрите шаблон спецификации требований • Определите классы пользователей • Смоделируйте среду приложения • Определите источники требований • Определите базовую и контрольную версии наборов требований • Определите лиц, ответственных за принятие решений по требованиям

Табл. 3-2. (*окончание*)

Влияние	Сложность		
	Высокая	**Средняя**	**Низкая**
Среднее	• Создайте матрицу связей требований • Проведите семинары по выявлениюю требований • Оцените объем работ по реализации требований • Повторно задействуйте существующие требования	• Ознакомьте представителей пользователей и менеджеров с требованиями • Проведите фокус-группы • Создайте прототипы • Проанализируйте осуществимость • Определите критерии приемлемости • Смоделируйте требования • Проанализируйте интерфейсы • Проанализируйте влияние, оказываемое изменениями • Выберите соответствующий цикл разработки проекта • Определите системные события и реакцию на них • Управляйте рисками, касающимися требований • Делайте выводы из полученного опыта • Отслеживайте объем работ по реализации требований	• Создайте словарь данных • Наблюдайте за пользователями на рабочих местах • Протестируйте требования • Отслеживайте состояние требований • Выполните анализ документов • Отлеживайте проблемы с требованиями • Задайте каждому требованию уникальный идентификатор • Создайте словарь терминов
Низкое		• Раздайте опросные листы • Отслеживайте хронологию изменений • Смоделируйте требования	• Изучите отчеты о проблемах

Не пытайтесь применять в своем следующем проекте сразу все эти приемы — их стоит рассматривать как новые компоненты вашего инструментария для работы над требованиями. Некоторые приемы, например касающиеся управления изменениями, можно начать использовать независимо от того, на какой стадии разработки находится ваш проект. Приемы выявления требований наиболее полезны в начале работы над очередным проектом или при повторении. Прочие способы могут оказаться неприемлемыми из-за ограничений текущего проекта, организационной культуры или объема доступных ресурсов. В главе 31 и Приложении А рассказывается, как оценить способы

создания требований, используемые вами в настоящий момент. Описанные в главе 31 приемы помогут вам разработать план совершенствования выявления и документирования требований. Внедрите в ваш корпоративный процесс улучшения ПО новые приемы работы с требованиями, полагаясь на руководство людей ответственных за изменения для организации пилотного внедрения, развертывания и принятия более совершенных приемов. Просто следите, чтобы ваши команды разработчиков при каждой возможности пробовали что-то новое и лучшее.

Что дальше?

- Вернитесь к проблемам с требованиями, которые вы определили, выполняя задания раздела «Что теперь?» главы 1. Выберите описанные в этой главе приемы, которые помогут вам решить все эти проблемы. Сгруппируйте приемы по их влиянию на вашу организацию (сильное, среднее и слабое). Определите, какие препятствия в организации или культуре способны помешать вам воспользоваться тем или иным приемом. Кто поможет вам устранить эти препятствия? Можете ли вы выбрать одно действие, чтобы работать эффективнее, чем раньше?
- Определите, как бы вы оценивали приемы, которые считаете наиболее ценными. Позволят ли они уменьшить число некорректных требований, выявляемых на поздних стадиях работы над проектом, уменьшить объем ненужных доработок, точнее следовать графику проекта или предоставят какие-то другие преимущества?
- Перечислите все приемы формулирования требований, определенные на первом этапе. Укажите, насколько члены вашей команды овладели на сегодняшний момент тем или иным приемом: эксперты, опытные специалисты, новички или не знакомые с данным приемом. Если квалификация членов команды ниже, чем опытных специалистов, попросите одного из них изучить данный прием и поделиться полученными знаниями с остальными членами команды.

Глава 4

Бизнес-аналитик

Молли — старший бизнес-аналитик в страховой компании, где работает уже семь лет. Менеджер недавно сообщил ей, что наблюдая за ее успешной работой на протяжении этого времени, он принял решение поручить ей обучение и найм персонала в отдел. Он спросил у Молли насчет идей, как должна выглядеть процедура найма новых бизнес-аналитиков и как нужно обучать уже имеющихся аналитиков. Молли была польщена. Она задумалась о собственном пути, чтобы посмотреть, не будет ли этот опыт полезным для совершенствования сотрудников отдела.

Молли защитила диплом по вычислительным системам, но у нее не было курса по работе с требованиями — основное ударение было сделано на техническую сторону разработки ПО. В начале карьеры она работала разработчиком корпоративного ПО. В течение года она поняла, что эта работа не совсем ей подходит. Молли проводила большую часть своего времени на рабочем месте и писала код, постоянно борясь со стремлением общаться с людьми. В течение следующих нескольких лет она выросла до роли бизнес-аналитика, хотя формально она все еще числилась разработчиком. В конце концов она убедила менеджера дать ее должности более соответствующее реалиям название и формально переопределить ее роль. Молли также прошла курс по работе с требованиями к ПО, чтобы получить фундаментальные знания. После этого она добилась назначения в проекты, где она смогла испытать на практике различные приемы и поучиться у более опытных специалистов. В течение следующих нескольких лет она смогла разработать процесс работы с требованиями в компании. Молли стала постоянным экспертом по бизнес-анализу.

Молли понимает, что нанимая новых бизнес-аналитиков, не следует ожидать наличия у них специализированного образования. При собеседовании с кандидатами на эту должность нужно проверять наличие самых важных коммуникативных навыков бизнес-аналитика. В ее программе обучения основной упор будет сделан на основах бизнес-анализа и на том, как применять важные коммуникативные навыки. Наконец, она создаст программу наставничества для младших бизнес-аналитиков.

Среди участников любого проекта по разработке ПО обязательно есть человек, явно или неявно выполняющий роль бизнес-аналитика. Бизнес-

аналитик делает возможными изменения в организационном контексте, определяя потребности и рекомендуя решения, которые представляют ценность для заинтересованных лиц. Задача аналитика — собрать и проанализировать мнения заинтересованных сторон и лиц, отразить их в спецификации требований и передать информацию другим заинтересованным в проекте лицам. Аналитик помогает участникам проекта прояснить, действительно ли пожелания, которые они высказывают вслух, — это то, что им на самом деле нужно. Аналитик обучает, задает вопросы, слушает, организует и учится. Это сложная работа.

В этой главе я познакомлю вас с неотъемлемыми функциями аналитика, требованиями, которые предъявляются к его знаниям и опыту, а также расскажу, как воспитать аналитика в своей организации (Wiegers, 2000; IIBA, 2011). Ральф Янг (Ralph Young, 2004) предлагает описание должности аналитика, кроме того пример описания должности бизнес-аналитика вы найдете в прилагаемом к этой книге материале.

Роль бизнес-аналитика

Бизнес-аналитик — это основное лицо, отвечающее за выявление, анализ, документирование и проверку требований к проекту. Это основной коммуникативный канал между группой клиентов и командой разработчиков (рис. 4-1), хотя, конечно, не единственный: есть и другие. Аналитик отвечает за сбор и распространение информации *о продукте*, а менеджер проекта — за обмен информацией *о проекте*.

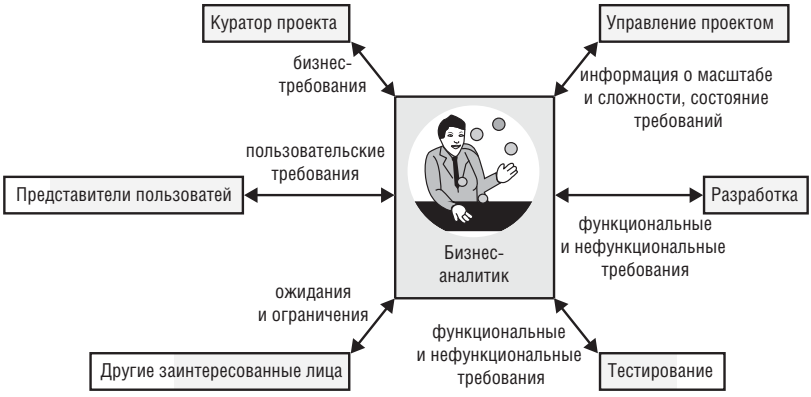


Рис. 4-1. Обязанности аналитика требований: наведение коммуникативных мостов между клиентом и разработчиками

Бизнес-аналитик — это одна из ролей участников проекта, а не обязательно название должности. Их также называют аналитиками по требованиям, системными аналитиками, инженерами по требованиям, менеджерами по требованиям, прикладными аналитиками, аналитиками бизнес-систем, ИТ-аналитиками и просто аналитиками. Название этой должности сильно раз-

нится в разных организациях. На эту роль можно назначить одного или нескольких специалистов. Кроме того, функции аналитика могут выполнять другие члены команды параллельно со своими обязанностями, например менеджер проекта, менеджер продукта, владелец продукта, специалист предметной области, разработчик и даже пользователь.

Важно заметить, что когда у человека, выполняющего роль бизнес-аналитика, есть другая роль в проекте, он выполняет две разных работы. Представьте, что менеджер проекта одновременно является бизнес-аналитиком в нем. Менеджер должен создавать и управлять планами, в том числе графиком и ресурсами, основываясь на работе, выполненной бизнес-аналитиком. Менеджер проекта должен помогать управлять границами проекта и корректировать график по мере изменения границ проекта. Сейчас он может выполнять роль менеджера проекта, а через минуту он будет выполнять работу бизнес-аналитика. Но это разные роли, требующие немного отличающихся наборов навыков.

В организациях, разрабатывающих потребительские продукты, роль аналитика часто выполняют менеджер продукта или специалисты по маркетингу. В сущности, менеджер продукта действует как бизнес-аналитик, часто с дополнительным ударением на понимании рыночного ландшафта и предугадывании потребностей внешних пользователей. Если в проекте есть и менеджер продукта, и бизнес-аналитик, первый обычно фокусируется на внешнем рынке и запросах пользователей, а второй преобразовывает эту информацию в функциональные требования.

В проектах гибкой разработки (agile) также нужен бизнес-анализ. В таких проектах обычно имеется роль владельца продукта, который выполняет часть традиционных задач бизнес-аналитика. А в некоторых командах предпочитают иметь также роль аналитика (Cohn, 2010). Бизнес-аналитик помогает представить пользователей и понять их потребности, а также выполняет дополнительные действия бизнес-аналитика, описанные далее в этой главе. Независимо от названия должности, человек, выполняющий задачи аналитика, должен обладать соответствующими навыками, знанием и личными качествами.

> **Внимание!** Не думайте, что любой талантливый разработчик или опытный пользователь автоматически, без обучения, чтения литературы и тренировок, станет профессиональным аналитиком требований. Все эти роли требуют разных навыков, знаний и личных качеств.

От таланта аналитика зависит успех проекта. Один из клиентов, которых я консультировал, пришел к выводу, что спецификации с требованиями, созданные опытными аналитиками, удается изучать вдвое быстрее, чем созданные новичками, поскольку в первых меньше недостатков. В модели Cocomo II, широко применяемой для оценки проектов, опыт и способности аналитика требований сильно влияют на материальные и трудовые затраты, связанные с реализацией проекта (Boemn et al., 2000). Привлекая опытных аналитиков,

можно на треть снизить связанные с проектом трудозатраты по сравнению с аналогичными проектами, где заняты неопытные аналитики.

Задачи аналитика

Задача аналитика — прежде всего выяснить, для чего нужна пользователям новая система, и затем определить пользовательские, функциональные и качественные требования, на основе которых команды смогут оценить и спланировать проект, а также спроектировать, построить и проверить продукт. Аналитик — это посредник в общении, проясняющий смутные представления пользователей и обращающий их в четкие спецификации, которыми руководствуется команда разработчиков продукта. В этом разделе описаны некоторые стандартные обязанности аналитика.

Определить бизнес-требования Ваша работа в качестве аналитика начинается, когда вы помогаете куратору, менеджеру продукта или менеджеру по маркетингу определить бизнес-требования к проекту. Можно разработать шаблон документа о концепции и границах (см. главу 5) и, расспросив людей, имеющих представление о концепции системы, получить у них необходимую информацию.

Спланировать подход к работе с требованиями Аналитик должен разработать планы выявления, анализа, документирования, проверки и управления требованиям на всем протяжении проекта и тесно сотрудничать с менеджером проекта над согласованием этих планов с общим планом проекта для достижения поставленных целей.

Определить заинтересованных лиц и классы пользователей Совместно с кураторами следует выбрать соответствующих представителей каждого класса (см. главу 6), заручиться их поддержкой и согласовать обязанности. Запишите, какого именно сотрудничества вы ожидаете от пользователей и согласуйте уровень участия каждого из них.

Выявить требования Профессиональный аналитик помогает пользователям четко обрисовать функции системы, необходимые им для достижения бизнес-целей, используя приемы сбора информации. Подробнее см. главы 7 и 8.

Анализировать требования Аналитик должен искать производные требования, логически проистекающие из запросов клиентов, а также невысказанные ожидания, которые, как считают клиенты, и так будут реализованы. Он использует модели требований, чтобы выявить паттерны, пробелы в требованиях, конфликтующие требования и убедиться, что все требования укладываются в границы проекта. Аналитик работает с заинтересованными лицами, чтобы определить пользовательские и функциональные требования с нужной степенью детализации.

Документировать требования Аналитик отвечает за создание хорошо организованного и написанного документа требований, который описывает решение, удовлетворяющее потребности клиента. Применение стандартных шаблонов ускоряет разработку требований, поскольку у аналитика перед

глазами будет постоянно находиться перечень тем, которые нужно обсудить с представителями пользователей.

Доводить требования до заинтересованных лиц Аналитик должен эффективно довести требования до всех участников. Аналитик должен определить, полезно ли представлять требования не текстовыми средствами, например с помощью разнообразных моделей графического анализа (см. главы 5, 12 и 13), таблиц, математических уравнений, раскадровок и прототипов (см. главу 15). Доведение требований не ограничивается распечаткой схем и развешиванием их на стенах — это постоянное взаимодействие с командой, чтобы быть уверенным в правильности понимания доводимой аналитиком информации.

Управлять проверкой требований Аналитик должен гарантировать, что формулировки требований отвечают всем характеристикам, перечисленным в главе 11, и что система, созданная на основе этих требований, устроит пользователей. Аналитики — ключевые участники рецензирования документов с требованиями. Им также следует изучить архитектуру, код и варианты тестирования, спроектированные на основе спецификаций требований, и убедиться, что требования интерпретированы правильно. Если аналитик создает приемочные тесты вместо подробных требований в проектах гибкой разработки, эти тесты тоже должны подвергаться рецензированию.

Обеспечить расстановку приоритетов требований Аналитик обеспечивает общение и взаимодействие различных классов заинтересованных лиц с разработчиками, чтобы расставить приоритеты требований в соответствии с бизнес-целями.

Управлять требованиями Бизнес-аналитик вовлечен во все этапы разработки ПО, его задача — помочь создать, обсудить и осуществить план управления требованиями к проекту. Определив базовую версию требований для текущего выпуска продукта или итерации разработки, бизнес-аналитик переходит к отслеживанию состояния этих требований, проверкой того, как они реализуются в продукте и управлением изменениями базовых требований. Расспрашивая коллег, аналитик собирает информацию о связях требований, сопоставляет их с прочими элементами системы.

Навыки, необходимые аналитику

Не думайте, что без достаточной подготовки, тренировок и опыта человек сможет стать аналитиком. Ему не только не удастся хорошо выполнять работу, он просто разочаруется в ней. Эта должность предусматривает наличие «социальных навыков», которые более ориентированы на работу с людьми, чем на технические аспекты. Аналитик должен уметь применять разные средства сбора информации и представлять эту информацию различными способами на нормальном и понятном языке. Профессионал в этой области обладает одновременно развитыми коммуникационными навыками, знанием психологии межличностного общения, техническими знаниями, знаниями предметной области бизнеса и личными качествами, подходящими для

этой работы. Основные факторы успеха — терпение и искреннее желание работать с людьми. Особенно важны навыки, описанные в этом разделе. Янг (Young, 2004) предоставляет исчерпывающую таблицу навыков аналитиков требований начального, среднего и старшего уровня.

Умение слушать Чтобы стать специалистом, аналитик должен научиться эффективно слушать. Активное слушание подразумевает устранение помех, сохранение вежливой позы и зрительный контакт, а также повторение основных моментов для закрепления их понимания. Нужно моментально схватывать, что говорят люди, и уметь читать между строк, чтобы обнаружить вещи, о которых они стесняются говорить. Аналитик должен изучить, как коллеги предпочитают общаться и избегать налагать собственный фильтр понимания на высказывания клиентов. Нужно искать допущения, которые подчеркивали бы мысли других или их личную интерпретацию.

Умение опрашивать и задавать вопросы Большая часть информации о требованиях извлекается в ходе бесед с людьми, поэтому аналитик должен уметь общаться с разными людьми и группами — только так ему удастся выявить их потребности. Возможно, работать со старшими менеджерами или чрезмерно самоуверенными или агрессивными людьми будет трудно. Для выяснения существенных требований пользователей необходимо задавать правильные вопросы. Например, пользователи обычно делают акцент на ожидаемом поведении системы. Тем не менее, значительная часть кода будет обрабатывать исключения, поэтому следует определить возможные условия ошибок и реакцию системы на них. По мере приобретения опыта аналитик учится задавать вопросы, которые раскрывают и проясняют неопределенности, расхождения во мнениях, предположения и невысказанные ожидания (Gause и Weinberg, 1989).

Способность соображать на ходу Бизнес-аналитики всегда находятся в курсе существующей информации и обрабатывают на ее основе новую информацию. Они должны выявлять противоречия, неясности, неопределенность и допущения и своевременно обсуждать их. Можно создать идеальный список вопросов для интервью, но всегда приходится задавать дополнительные вопросы. Нужно записать хорошие вопросы, внимательно прислушаться к ответам и быстро задать следующий самый точный в данной ситуации вопрос. Иногда нужно не задавать вопрос, а представить пример в контексте, который позволит заинтересованным лицам сформулировать следующий вопрос.

Навыки анализа Эффективный бизнес-аналитик способен думать на нескольких уровнях абстракции и знать, когда переходить между ними. Иногда требуется перейти от сведений высшего порядка к подробностям. В некоторых случаях на основе потребности одного из пользователей надо обобщить набор требований, которые удовлетворят большинство заинтересованных лиц. Бизнес-аналитик должен понимать сложную информацию, поступающую из многих источников, и решать непростые задачи, связанные с ней. Бизнес-аналитик критически оценивает информацию, полученную на основе разных источников, чтобы урегулировать конфликты, отделить мимолетные

желания пользователей от их реальных потребностей и отличать варианты решений от требований.

Навыки системного мышления Ориентируясь на детали, аналитик обязан не упускать общей картины и проверять требования на основе своих знаний о предприятии в целом, бизнес-среде и приложении, чтобы вовремя обнаружить несоответствия и влияние различных компонентов. Аналитик должен понимать взаимодействие и связи между людьми, процессами и технологиями, относящимися к системе (IIBA, 2009). Когда клиент запрашивает требование для своей функциональной области, бизнес-аналитик должен оценить, не повлияет ли это требование на другие подсистемы каким-то неожиданным образом.

Навыки обучения Аналитики должны быстро усваивать новый материал, будь то новые способы работы с требованиями или особенности предметной области. Аналитик должен отлично владеть языком и ясно выражать сложные идеи. Аналитик же должен уметь читать критично и эффективно, поскольку ему приходится просматривать множество материалов и необходимо быстро уяснять их суть. Не обязательно быть специалистом в предметной области — надо просто не стесняться задавать уточняющие вопросы. Аналитик должен честно признавать, что он чего-то не знает. Нет греха в том, что аналитик знает не все, но очень плохо, когда он скрывает отсутствие знаний.

Навыки создания комфортных условий общения Умение организовывать дружескую атмосферу на семинарах для уточнения требований — один из необходимых навыков аналитика. Создание доверительных отношений позволяет вести группу к успеху. Оно жизненно необходимо при коллективном определении требований, приоритизации потребностей и разрешении конфликтов. Нейтральный наблюдатель, имеющий опыт опроса, наблюдения и создания комфортных условий общения, создаст доверительные отношения в группе и уменьшит напряженность в отношениях между бизнес-пользователями и ИТ-сотрудниками. В главе 7 даны некоторые рекомендации по организации семинаров.

Лидерские качества Сильный аналитик может подталкивать группу заинтересованных лиц в определенном направлении для достижения общей цели. Лидерство подразумевает набор приемов достижения согласия между заинтересованными лицами проекта, разрешения конфликтов и принятия решений. Аналитик должен формировать атмосферу сотрудничества и доверия между разными группами заинтересованных лиц, которые могут не понимать мотивации, потребностей и ограничений своих коллег.

Умение наблюдать Внимательный аналитик запомнит высказанные мимоходом комментарии, которые могут оказаться важными. Наблюдая за тем, как пользователь выполняет свои обязанности или работает с имеющимся приложением, опытный аналитик выявит моменты, о которых пользователь даже не упомянул. Наблюдательность иногда помогает направить дискуссию в новое русло, чтобы выявить дополнительные требования, о которых никто не упоминал.

Навыки общения Основной итог процесса создания требований — письменная спецификация с информацией для клиентов, отдела маркетинга и технического персонала. Аналитик должен отлично владеть языком и ясно выражать сложные идеи как письменной, так и устной форме. Он должен уметь писать для разной аудитории, в том числе для клиентов, которые должны проверять требования, и для разработчиков, которым нужны очень четкие и точные формулировки для реализации. Бизнес-аналитик должен говорить ясно, подстраиваясь к местной терминологии и региональным различиям в диалекте. Также он обязан уметь суммировать и представлять информацию с уровнем детализации, который нужен целевой аудитории.

Организационные навыки Аналитик имеет дело с большим объемом беспорядочной информации, собранной на этапе выявления и анализа. Чтобы справиться с данными и выстроить согласованное целое, вам потребуются исключительные организационные навыки, а также терпение и упорство для вычленения основных идей из хаоса. Аналитик должен уметь создавать архитектуру информации, поддерживающую информацию проекта по мере его продвижения (Beatty и Chen, 2012).

Навыки моделирования Аналитик должен уметь работать с разнообразными средствами, начиная с древних блок-схем и структурированных моделей анализа (диаграммы потоков информации, диаграммы «сущность-связь» и т. д.) и заканчивая современным языком UML (Unified Modeling Language) (Beatty and Chen, 2012). Некоторые из этих средств полезны при общении с пользователями, другие — с разработчиками, но есть и предназначенные исключительно аналитику для облегчения работы с требованиями. Бизнес-аналитик должен знать, как выбирать модели на основе ценности, которую они могут принести. Аналитику следует объяснить другим участникам проекта ценность использования этих методов и то, как работать с их данными. Обзор некоторых типов моделей анализа см. в главах 5, 12 и 13.

Навыки межличностного общения Аналитик должен уметь организовать людей с разными интересами для совместной работы, и уверенно чувствовать себя в разговорах с сотрудниками, занимающими разные должности в организации. Бизнес-аналитик должен разговаривать на языке целевой аудитории, не прибегая к техническому жаргону при общении с заинтересованными лицами. Он также должен уметь работать с сотрудниками из виртуальных групп, различающихся по географическому, временному, культурному или языковому признаку. Бизнес-аналитик также должен уметь легко общаться и понятно формулировать свои мысли в разговорах с членами команды.

Творческий подход Аналитик — не просто клерк, записывающий все высказывания клиентов. Лучшие аналитики изобретают требования (Robertson, 2002). Они предлагают инновационные функции продуктов, новые рыночные возможности и возможности для бизнеса и думают, как удивить и удовлетворить своих клиентов. Отличный аналитик творчески подходит к делу: рассказывая о системе, ему удается удивить клиента — тот даже не всегда подозревает, что такая функциональность возможна. Аналитики могут предлагать новые

идеи, потому что в отличие от пользователей видят проблему с расстояния. Но аналитики не должны навешивать лишние «бантики» на решение — новые требования не добавляются в спецификацию без одобрения клиента.

Проверяйте на практике то, чему учите других

Один опытный бизнес-аналитик и разработчик однажды спас меня от самого себя. Я разговаривал со своей коллегой Таней о программном сервисе, который я считал нужно создать на моем веб-сайте. Я сказал ей, что мне нужен какой-то сценарий, который будет перехватывать определенные сообщения электронной почты, поступающие на мой адрес, и анализировать определенную информацию в них. Я не знал, как написать такой сценарий, поэтому я спросил у Тани, как мне поступить.

Таня ответила: «Извини меня, Карл, но я не думаю, что это твое настоящее требование. На самом деле ты хочешь получить нужную тебе информацию способом, отличным от ручного чтения и обработки поступающих в твой почтовый ящик сообщений». Она была совершенно права. Я попался в очень популярную ловушку, когда пользователь пытается определить решение как требование. К счастью, наблюдательный бизнес-аналитик обнаружил мою ошибку. Таня сделала шаг назад и моментально ухватила базовую проблему. После этого практически всегда становится понятным, что есть несколько способов ее решения, часть из которых может быть лучше того, что пришло в голову первым. Мой сообразительный друг Таня напомнила мне, как важно для квалифицированного бизнес-аналитика копать чуть глубже представленного решения и понимать настоящие цели пользователя.

Знания, необходимые аналитику

Помимо специальных навыков и личных качеств, бизнес-аналитик должен обладать обширными знаниями, большая часть которых приходит с опытом. Он должен понимать современные приемы работы с требованиями и знать, как их применять на различных стадиях разработки ПО. От него может потребоваться обучать и убеждать тех, кто не знаком с принятыми приемами работы с требованиями. У опытного аналитика в арсенале большой набор приемов, и он знает, когда надо — и когда не стоит — их применять.

Бизнес-аналитик должен проводить линию разработки и управления требованиями на протяжении всего жизненного цикла проекта. Аналитик с хорошим пониманием управления проектами, моделей разработки, управления рисками и техники управления качеством может защитить проект от проблем, возникающих из-за неправильной работы с требованиями. При коммерческой разработке бизнес-аналитику будет полезно знание принципов управления продуктом. Бизнес-аналитику также нужны базовые знания об архитектуре и эксплуатационных условиях, чтобы он мог вести разговор о приоритетах и нефункциональных требованиях.

Знание о бизнесе, отрасли и организации — мощные активы эффективного бизнес-аналитика (IIBA, 2009). Владение знаниями о бизнесе позволяет аналитику свести к минимуму недопонимание с пользователями. Аналитикам, понимающим организацию и предметные области, часто удается обнаруживать невысказанные предположения и неявные требования. Они могут предложить пользователям способы улучшения их бизнес-процессов или ценную функциональность, о которой другие заинтересованные лица не догадались. Понимание отрасли может оказаться особенно полезным при коммерческой разработке, где аналитики могут предложить анализ рынка и конкурентных преимуществ продукта.

Становление аналитика

В великие аналитики приходят из разных профессий, и, скорее всего, у всех новичков есть пробелы в знаниях и навыках. Тому, кто собирается заниматься этим делом, следует определить, какие именно из обсуждаемых в этой главе требований относятся к нему, и постараться активно восполнить пробел, чтобы первоклассно выполнить работу. Международный институт бизнес-анализа (International Institute of Business Analysis, IIBA) описывает требования к профессиональному уровню начинающего, опытного и ведущего аналитика требований в разных областях (IIBA, 2011). Аналитику-новичку пригодятся советы и наставления опытных коллег, выраженные, скажем, в форме обучения. Давайте посмотрим, как люди с разным профессиональным опытом становятся аналитиками.

Бывший пользователь

Во многих корпоративных отделах информационных технологий есть сотрудники, пришедшие в бизнес-аналитики из обычных пользователей. Они отлично понимают особенности бизнеса и рабочей среды и легко завоевывают доверие бывших коллег. Они знают язык пользователей, а также существующие системы и бизнес-процессы.

К сожалению, бывшие пользователи зачастую имеют весьма поверхностные знания о разработке ПО и взаимодействии с техническими специалистами. Если они не знакомы с методами моделирования анализа, то по привычке выражают всю информацию в текстовой форме. Пользователям, ставшим бизнес-аналитиками, следует побольше узнать о технической стороне разработки ПО, чтобы представлять информацию в наиболее подходящей для разных аудиторий форме.

Некоторые бывшие пользователи считают, что они лучше, чем те, кто работает с ПО теперь, понимают, что на самом деле необходимо, и поэтому не обращаются к этим сотрудниками или неуважительно относятся к информации, предоставленной теми, кто будет работать с новой системой. Недавние пользователи могут запутаться в текущих особенностях работы, да так, что

не увидят возможности усовершенствовать бизнес-процессы посредством новой информационной системы. Еще одна опасность: бывший пользователь может запросто думать о требованиях только с точки зрения пользовательского интерфейса. Концентрация внимания на вариантах решения с самого начала создает ненужные ограничения дизайна и зачастую не позволяет решить реальную проблему.

От специалиста по медицинской технике к аналитику требований

У старшего менеджера отдела медицинских устройств в большой компании возникла проблема. «Два года назад я принял на работу в свой отдел трех специалистов по медицинскому оборудованию, чтобы те представляли потребности наших клиентов. Они проделали огромную работу, но уже не владеют особенностями современных медицинских технологий и поэтому не могут точно сказать, что же сегодня нужно клиентам. Какую работу мы можем предложить этим специалистам теперь?»

Бывшие специалисты по медицинскому оборудованию, работающие под началом этого менеджера могут стать кандидатами в бизнес-аналитики. И хотя они не знают о новинках медицинской техники, они смогут общаться на одном языке с другими специалистами в этой области. За два года они усвоили принципы работы отдела. Возможно, им потребуется пройти обучение способам документирования требований, тем не менее, они накопили ценный опыт, который позволит им эффективно выполнять работу аналитиков. Эти бывшие пользователи успешно перешли к выполнению роли бизнес-аналитика.

Бывший разработчик или тестировщик

Менеджеры проекта, которым не хватает специализированного аналитика требований, зачастую ожидают, что его функции будет выполнять разработчик. К сожалению, навыки и личные качества, необходимые разработчику, отличаются от тех, что необходимы аналитику. Мало кто из разработчиков терпелив с пользователями, считая их необходимым злом, с которым нужно разделаться как можно быстрее, чтобы скорее вернуться к реальной работе — программированию. Конечно, многие разработчики осознают важность процесса создания требований и высказывают желание работать аналитиками, когда потребуется. Те, кому нравится общаться с пользователями — хорошие кандидаты для специализации в области бизнес-анализа.

Разработчику, ставшему аналитиком, скорее всего придется более подробно ознакомиться с предметной областью бизнеса. Однако разработчики легко переходят на техническое мышление и жаргон, концентрируясь вместо потребностей пользователей на программном продукте, который надо создать. Им придется обучиться новейшим приемам разработки требований.

Разработчикам будет полезно дополнительное обучение навыкам межличностных коммуникаций, которыми владеют аналитики-профессионалы.

Тестировщикам нечасто предлагают взять на себя роль аналитика. Однако у тестировщиков часто бывает аналитический склад ума, который подходит для работы бизнес-аналитиком. Ум тестировщиков уже настроен на обнаружение исключений и как система может ломаться — полезный навык для обнаружения пробелов в требованиях. Как и бывшему разработчику, тестировщику придется освоить правильные приемы разработки требований. Также может потребоваться более глубоко освоить предметную область.

Бывший (или текущий) менеджер проекта

Бывает, что роль бизнес-аналитика поручают менеджерам проекта в дополнение к основной работе — обычно потому, что у них есть такие же навыки и знания предметной области. Это может оказаться эффективной сменой роли. Менеджеры проектов уже работают с соответствующими командами, понимают организацию и предметную область и обладают хорошими навыками общения. Они наверняка умеют слушать, обладают навыками переговоров и координации работы. У них также отличные навыки организации и написания документов.

Однако бывший менеджер проекта должен больше узнать о приемах разработки требований. Одно дело создать план, выделить ресурсы и скоординировать действия аналитика и других членов команды, но совсем другое самому играть роль бизнес-аналитика. Бывший менеджер проекта должен научиться концентрироваться на понимании бизнес-потребностей и определении их приоритетов в рамках графика текущего проекта, а не на шкале времени, ресурсах и бюджетных ограничениях. Потребуется развить в себе навыки анализа, моделирования и интервьюирования, которые менее важны для менеджера проекта, но абсолютно необходимы для успеха бизнес-аналитика.

Специалист предметной области

Ральф Янг (Ralph Young, 2001) рекомендует, чтобы бизнес-аналитик был экспертом в предметной области или профильным специалистом, а не обычным пользователем: «Специалисты предметной области могут определить, насколько разумны требования, как они расширяют существующую систему, как следует проектировать предполагаемую архитектуру и какое влияние они окажут на пользователей». Некоторые организации-разработчики ПО нанимают опытных пользователей их продуктов, обладающих большим опытом в предметной области, в качестве аналитиков или представителей пользователей.

Здесь тоже есть свои риски. Бизнес-аналитик, будучи экспертом в предметной области, зачастую определяет требования к системе, которые соответствуют его личным предпочтениям, а не обоснованным потребностям различных классов пользователей. У него может быть зашоренный взгляд на требования и пониженная способность генерировать новые идеи. Специалисты предметной области являются экспертами в понимании системы, которая

есть, и часто испытывают сложности с воображением, какой система должна быть. Зачастую лучше, чтобы бизнес-аналитик из команды разработчиков взаимодействовал с профильным специалистом, который кроме того выбран в качестве ключевого представителя пользователей или сторонника продукта. Подробнее о сторонниках продукта см. главу 6.

Молодой специалист

Стать бизнес-аналитиком — хорошее начало карьеры в информационных технологиях для человека, только что закончившего учебное заведение или пришедшего из совершенно другой области. У выпускника ВУЗа вообще нет или очень маленький объем знаний и опыта. На должность бизнес-аналитика его скорее всего взяли из-за того, что он демонстрирует многие из навыков, которые нужны хорошему аналитику. Преимущество найма новичка в качестве бизнес-аналитика в том, что у него нет закостенелых понятий о том, как выполняются процессы работы с требованиями.

Из-за недостатка опыта и знаний выпускнику придется много изучать особенности работы и приемов бизнес-аналитика. Недавний выпускник должен также изучить процесс разработки ПО, чтобы понимать проблемы разработчиков, тестировщиков и других членов команды — это поможет ему эффективно сотрудничать с ними. Наставник может сократить время вхождения новичка-аналитика в профессию и с самого начала привить хорошие привычки.

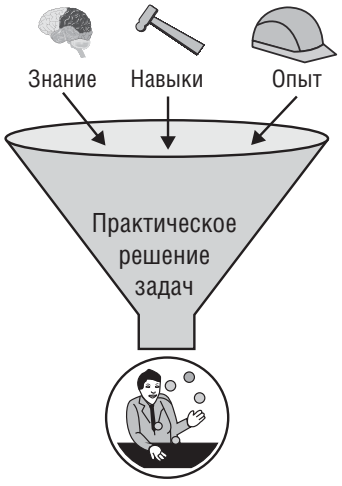


Рис. 4-2. Знания, навыки и опыт — основа для формирования эффективного бизнес-аналитика

Изобретательный бизнес-аналитик всегда найдет способ применить свой опыт, каким бы он ни был до этого. Аналитик должен освоить знания и навыки, которых ему пока не хватает, задействовать свой прошлый опыт, а также практиковаться в выполнении реальных аналитических задач бизнес-

аналитика. Все это позволяет сформировать полноценного бизнес-аналитика (рис. 4-2).

Роль аналитика в проектах гибкой разработки

В проектах *гибкой разработки* (agile) также нужны функции бизнес-аналитика, но соответствующая должность может называться иначе. В некоторых вариантах гибкой разработки есть ключевой член команды, который называется *владелец продукта* (product owner). Человек с этой ролью может выполнять традиционный бизнес-анализ, а также создавать концепции продукта, доводить ограничения, определять приоритеты *резерва* (backlog) проекта и принимать конечные решения по продукту (Cohn, 2010). В других проектах роли бизнес-аналитика и владельца продукта разнесены. Дополнительно части роли аналитика могут выполнять другие члены команды, например разработчики. Суть в том, что задачи роли бизнес-аналитика должны выполняться независимо от способа реализации проекта. Команда только выиграет, если в ней будут люди, обладающие навыками бизнес-аналитика.

Часто в организациях, переходящих на гибкую разработку, бизнес-аналитик перестает понимать, какую пользу он может принести проекту. В духе гибкой разработки аналитик должен выйти из стандартной роли «бизнес-аналитик» и заняться тем, что позволит выпустить успешный продукт. Эллен Готтесдинер (Ellen Gottesdiener, 2009) предлагает подробный список того, как деятельность бизнес-аналитика можно приспособить к среде гибкой разработки. Вот ряд предложений бизнес-аналитику, как применить свои навыки в проекте гибкой разработки:

- определите облегченный гибкий процесс работы с требованиями и видоизмените его, как того требует проект;
- позаботьтесь о том, чтобы объем документации требований был ровно тем, какой нужен, — не больше и не меньше (Многие бизнес-аналитики стремятся задокументировать в спецификации все, вплоть до энного уровня. Некоторые борцы за чистоту методики утверждают, что в проектах гибкой разработки должно быть мало или вообще не должно быть документации требований. Ни одна из этих крайних точек зрения не является идеальной.);
- помогите определить наилучший подход к документированию резерва, в том числе нужны ли карты историй или более формальные инструменты;
- используйте свои лидерские способности и навыки создания комфортных условий общения, чтобы обеспечить частое обсуждение заинтересованными лицами потребностей, вопросов и сложностей с требованиями;
- помогите проверить, все ли потребности клиента отражены в резерве продукта и организуйте приоритизацию требований в резерве;
- работайте с клиентами, когда они меняют свое мнение о требованиях и приоритетах, и помогите зафиксировать эти изменения. Работайте с другими

членами команды над определением влияния изменений на итерацию и планы выпуска.

Использование такой роли, как владелец продукта для преставления пользователей во время разработки, очень ценно для проекта, однако у человек, играющий эту роль, может не обладать всеми навыками бизнес-аналитика или временем для выполнения всех необходимых действий. Эти критически важные возможности может привнести в команду полноценный бизнес-аналитик.

Создание дружной команды

Иногда в проектах по разработке ПО возникают напряженные отношения между аналитиками, разработчиками, пользователями, менеджерами и специалистами по маркетингу. Стороны не всегда доверяют мотивации других участников и не ценят потребности и ограничения других. Однако на самом деле у разработчиков и потребителей программного продукта цель общая. При создании корпоративных информационных систем все стороны работают на одну компанию и все они получают выгоду от улучшения результатов работы компании. В случае успеха коммерческого продукта налицо счастливые клиенты, разбогатевший производитель и удовлетворенный разработчик.

Бизнес-аналитик — основное лицо, отвечающее за обеспечение тесного доброжелательного сотрудничества с представителями пользователей и прочими сторонами, заинтересованными в проекте. Эффективный аналитик принимает во внимание трудности, с которыми сталкиваются представители бизнеса и технических служб, и постоянно демонстрирует коллегам свое уважение. Аналитик добивается от участников проекта согласования требований, в результате чего выигрывают все:

- клиенты довольны продуктом;
- организация-разработчик получает прибыль;
- разработчики гордятся работой, выполненной в ходе сложного и стоившего того проекта.

Что дальше?

- Если вы бизнес-аналитик, сравните свои знания и навыки с описанными в этой главе, чтобы выявить области, которые нужно подтянуть. Для этой цели хорошо подойдет анкета самооценки IIBA (IIBA, 2010). Создайте личный план по восполнению пробелов в знаниях.
- Если вы выявили какие-либо пробелы, выберите две темы и проработайте их, читая, практикуясь, работая с наставником или посещая специализированные курсы.
- Оцените ваши текущие знания о бизнесе, отрасли и организации, в которой вы сейчас работаете и определите, какие опыт и знания вам нужно развить. Найдите статью или специалиста, которые помогут вам восполнить пробел.

Часть II

Разработка требований

Глава 5

Определение бизнес-требований

Карен — бизнес-аналитик в проекте по созданию нового интернет-каталога товаров для сотрудников клиентской службы заказчика. Предварительная версия спецификации требований (SRS) была уже на этапе рецензирования, когда менеджер по маркетингу сказал, что хочет добавить на страницы товара кнопку «Нравится». Первая реакция Карен была отрицательной, потому что она сомневалась, что удастся вписаться в имеющееся время даже с текущим набором требований. Но потом осознала, что это может оказаться очень удачной функцией, потому что она позволит сотрудникам клиентской службы продвигать наиболее популярные товары среди других клиентов. Прежде чем собирать и документировать функциональные требования для этой функции, ей необходимо объективно проанализировать, нужно ли добавлять эту функцию в рамки проекта.

Когда она попыталась объяснить менеджеру по маркетингу необходимость дополнительного анализа этого запроса, он ответил: «Но ведь разработчики все равно скоро начнут писать код. Разве сложно добавить еще одну маленькую функцию?» Проведенный Карен анализ показал, что предлагаемая функция не вписывается в границы проекта: она не способствует достижению бизнес-цели сократить среднее время звонка сотрудника клиентской службы и непроста в реализации. Карен нужно суметь ясно объяснить, почему функцию нельзя включить в границы проекта, менеджеру по маркетингу, который не совсем в курсе сформулированных бизнес-целей.

Как мы уже видели в главе 1, бизнес-требования представляют высший уровень абстракции в цепи требований: они определяют концепцию решения и границы проекта, в котором оно будет реализовываться. Пользовательские и функциональные требования к ПО должны находиться в соответствии с контекстом и целями, устанавливаемыми бизнес-требованиями. Требования, не содействующие достижению бизнес-целей проекта, реализовываться не должны.

Проект, в котором нет четко определенного и согласованного направления, можно смело назвать кандидатом на провал. Участники проекта могут, сами того не осознавая, решать прямо противоположные задачи, если у них разные бизнес-цели и приоритеты. Лица, заинтересованные в проекте, ни-

когда не смогут договориться о составе требований, если они не выработали общего понимания бизнес-целей. Без четкого понимания с самого начала график и бюджет проекта скорее всего выйдут за намеченные рамки.

В этой главе описывается документ концепции и границ — результирующий документ, содержащий бизнес-требования проекта. На рис. 5-3 далее в этой главе предлагается шаблон документа концепции и границ. Но прежде узнаем, что такое «бизнес-требования».

Формулировка бизнес-требований

Термин *бизнес-требования* (business requirements) относится к информации, которая в совокупности описывает потребность, которая инициирует один или больше проектов, призванных предоставить решение и получить требуемый конечный бизнес-результат. В основе бизнес-требований лежат бизнес-возможности, бизнес-цели, критерии успеха и положение о концепции.

Вопросы бизнес-требований должны решаться до окончательного определения функциональных и нефункциональных требований. Положение о рамках и ограничениях проекта сильно помогает в обсуждениях предлагаемых функций и целевых выпусков. Бизнес-требования являются отправной точкой для принятия решений о предложенных изменениях и улучшениях требований. Мы рекомендуем представлять бизнес-цели, концепцию и границы на всех семинарах по сбору требований, чтобы в команде могли быстро понять, находится ли предлагаемое требование в рамках проекта.

Определение требуемых бизнес-преимуществ

Бизнес-требования определяют контекст и позволяют измерять преимущества, которые организация ожидает получить от реализации проекта. Организации не должны инициировать проект без ясного понимания пользы, которую он принесет для бизнеса. Определяйте измеряемые ориентиры на основе бизнес-целей, после чего определяйте критерии успеха, который позволят оценивать, находитесь ли вы на пути достижения этих целей.

Бизнес-требования могут исходить от финансирующих проект заказчиков, топ-менеджеров, менеджеров по маркетингу или ответственных за концепцию продукта. Однако определить и донести бизнес-преимущества бывает непросто. Члены команды не всегда уверены в том, какова задача проекта. Иногда кураторы не хотят определять цели в поддающемся измерению виде, чтобы потом не нести ответственность за их достижение. Может быть несколько заинтересованных лиц, которые не согласны с целями. Бизнес-аналитик должен обеспечить, чтобы бизнес-требования определяли правильные заинтересованные лица и организовать сбор, приоритизацию и разрешение конфликтов. Карл Вигерс (Karl Wiegers, 2006) предлагает ряд вопросов, которые должен задавать бизнес-аналитик при сборе бизнес-требований.

Бизнес-преимущества должны представлять реальную пользу для кураторов и клиентов проекта. Например, слияние двух систем в одну не является разумной бизнес-целью. Клиентам все равно, используют ли они одну, пять или десять систем — их интересуют задачи, такие как повышение доходов или снижение издержек. Слияние двух систем может быть частью решения, но само по себе оно редко является бизнес-целью. У проектов по обеспечению выполнения требований регулирующих органов также есть ясные бизнес-цели. Часто цели формулируются как задача избежать рисков, например риска судебного преследования или прекращения работы компании.

Концепция продукта и границы проекта

Концепция и границы — два базовых элемента бизнес-требований. *Концепция продукта* (product vision) сжато описывает конечный продукт, который достигнет заданных бизнес-целей. Этот продукт может полностью удовлетворять бизнес-требования или быть только частью решения. Концепция описывает, что продукт представляет собой сейчас и каким он станет впоследствии. Она обеспечивает контекст для принятия решений на протяжении жизненного цикла продукта и выстраивает работу всех заинтересованных лиц в одном направлении. *Границы проекта* (project scope) показывают, на какую часть конечной концепции продукта будет направлен текущий проект или итерация. В положении о границах определена черта между тем, что входит в проект и тем, что остается вовне.

> **Внимание!** Концепция продукта гарантирует, что мы все знаем, куда идем. Границы проекта гарантируют, что мы говорим об одной и той же вещи в текущем проекте или итерации.

Убедитесь, что концепция решает задачу

На одном из учебных семинаров мы дали слушателям бизнес-задачу и соответствующую бизнес-цель. В процессе выполнения практического задания мы периодически предоставляли дополнительные подробности требований. На каждом этапе мы просили слушателей предложить решение задачи на основе имеющейся информации. К концу выполнения задания у слушателей были похожие идеи решения, но редко кто решал исходную задачу!

Примерно то же происходит в реальных проектах. У команды могут быть ясные цели, создается спецификация, выполняется разработка и тестирование, но на протяжении проекта никто не проводит проверку на предмет соответствия целям. Заинтересованное лицо может предложить «блестящую» новую функцию, которую хочет реализовать. Команда добавляет ее, потому что это кажется разумным и интересным. Но несколько месяцев спустя оказывается, что несмотря на наличие массы крутых функций система не решает исходной задачи.

Говоря о концепции, мы подразумеваем весь продукт. Он будет изменяться относительно медленно при определении со временем стратегии продукта или развитии бизнес-целей. Границы же относятся к определенному проекту или его итерации, в которых реализуются возможности продукта, как показано на рис. 5-1. Границы более динамичны, чем концепция, так как заинтересованные лица изменяют содержимое каждой версии в соответствии с ограничениями графика, бюджета, ресурсов и качества. Границы текущего выпуска должны быть четко определены, но границы будущих выпусков тем менее четко определены, чем более дальняя перспектива рассматривается. Задача команды состоит в управлении границами конкретного проекта разработки или улучшения как определенным подмножеством стратегической концепции продукта.

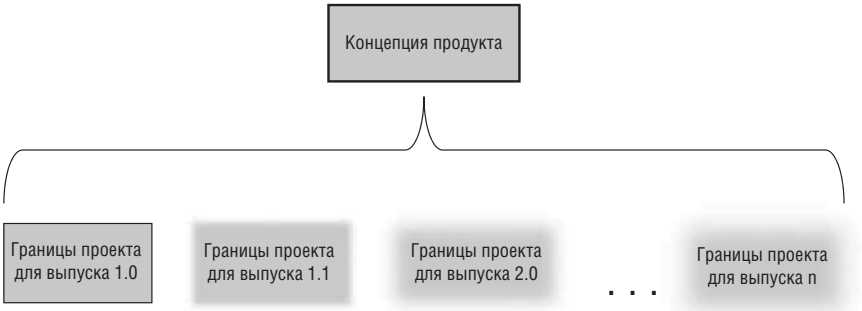


Рис. 5-1. Концепция продукта охватывает границы каждого запланированного выпуска и тем менее четко определена, чем более дальняя перспектива рассматривается

Перекрывающиеся границы

Федеральное агентство приступило к огромному, рассчитанному на пять лет проекту по разработке информационной системы. На раннем этапе процесса агентство определило бизнес-цели и концепцию системы, которые не будут существенно меняться на протяжении пяти лет. Запланировано 15 выпусков частей всей системы, каждая из которых создается отдельной проектной командой и у каждой есть собственное определение границы проекта. Некоторые проекты будут выполняться параллельно, потому что часть из них взаимосвязаны и время реализации различается. Каждое определение границ должно соответствовать общей концепции продукта и перекрываться с границами других проектов, чтобы ничего не упустить и четко определить рамки ответственности.

Противоречивые бизнес-требования

Бизнес-требования, собранные из многих источников, могут быть противоречивыми. Представьте себе интерактивный терминал, предназначенный для посетителей магазина. На рис. 5-2 показаны возможные бизнес-интересы

разработчика терминала, магазина и клиента, — как они себе представляют пользу, которую им может принести терминал.

Рис. 5-2. Бизнес-интересы заинтересованных в терминале лиц не всегда совпадают

Иногда интересы разных заинтересованных лиц совпадают. Например, разработчики и клиенты хотят, чтобы терминал предоставлял широкий выбор товаров и услуг. Однако некоторые бизнес-цели противоречат друг другу. Клиент хочет тратить меньше времени на совершение покупки, а розничная компания хочет, чтобы клиент проводил больше времени в магазине и тратил больше денег. Трения между заинтересованными лицами с разными целями и ограничениями ведут к конфликтующим бизнес-требованиям. Лица, ответственные за принятие решений, должны разрешить эти конфликты до того, как аналитик создаст подробные требования к терминалу. Основное ударение должно быть сделано на предоставлении максимальных бизнес-преимуществ основным заинтересованными лицам. Легко отвлечься на внешние характеристики продукта, которые не соответствуют поставленным бизнес-целям.

Лица, ответственные за принятие решений, не должны считать, что разрешением конфликтов между разными заинтересованными лицами будет заниматься команда разработчиков ПО. Рамки расширяются по мере увеличения круга заинтересованных лиц. Неконтролируемое распухание границ проекта, в котором заинтересованные лица требуют слишком многих функций от новой системы, желая удовлетворить все свои интересы, может привести к тому, что проект рухнет под собственным весом. На выручку может прийти бизнес-аналитик, который выявит области возможных конфликтов и различий в предположениях, отметит противоречивые бизнес-цели, отметит функции, не соответствующие этим целям и организует процесс разрешения

конфликтов. При разрешении подобных противоречий вступают в игру политика и сферы влияния, но эта книга не об этом.

В длительных проектах часто происходит смена ответственных за принятие решений. В этом случае нужно немедленно уточнить базовые требования с новыми ответственными. Они должны быть в курсе существующих бизнес-требований, потому что могут захотеть изменить их. Если это так, менеджеру проекта потребуется скорректировать бюджет, график и ресурсы, а бизнес-аналитику может понадобиться поработать с заинтересованными лицами над обновлением пользовательских и функциональных требований и их приоритетов.

Документ о концепции и границах

Документ о концепции и границах (vision and scope document) собирает бизнес-требования в единый документ, который подготавливает основу для последующей разработки продукта. В некоторых организациях с этой же целью создают устав проекта (Wiegers, 2007) или положение о бизнес-задачах. В организациях, создающих ПО на продажу часто создают *документ основных рыночных требований* (market requirements document, MRD). В нем более детально, чем в документе о концепции и границах, рассматриваются целевые сегменты рынка и вопросы коммерческого успеха продукта.

Владельцем документа о концепции и границах считается куратор проекта, тот, кто финансирует проект, или имеет аналогичную ответственность. Бизнес-аналитик может вместе с этим человеком разрабатывать документ о концепции и границах. Информация, касающаяся бизнес-требований, должна поступать от лиц, четко понимающих, почему они взялись за проект. Это может быть клиент или топ-менеджер организации, разрабатывающей продукт, ответственный за концепцию, менеджер по продукту, эксперт предметной области или специалисты отдела маркетинга.

На рис. 5-3 показан шаблон документа о концепции и границах, а далее в главе элементы шаблона описываются более детально. Как и в случае с любым шаблоном, этот нужно изменить в соответствии со спецификой проекта. Если вы уже зафиксировали информацию в других документах, не нужно дублировать ее в документе о концепции и границах. Некоторые части документа о концепции и границах повторяются от проекта к проекту, в их числе бизнес-цели, бизнес-риски и профили заинтересованных лиц. В Приложении В приводится пример документа о концепции и границах, созданного на основе этого шаблона.

Документ концепции и границ определяет границы на высоком уровне, а подробности границ представлены базовыми требованиями отдельных выпусков, определенных командой. Крупные новые проекты должны иметь завершенный документ концепции и границ и спецификацию требований. (Подробнее о шаблоне SRS см. главу 10.) Каждая итерация, выпуск или проект по улучшению продукта может включать собственное положение о гра-

ницах в документации требований к проекту, при этом создавать отдельный документ концепции и границ необязательно.

1. Бизнес-требования
   1.1 Исходные данные
   1.2 Возможности бизнеса
   1.3 Бизнес-цели
   1.4 Критерии успеха
   1.5 Положение о концепции проекта
   1.6 Бизнес-риски
   1.7 Предположения и зависимости
2. Рамки и ограничения проекта
   2.1 Основные функции
   2.2 Объем первоначально запланированной версии
   2.3 Объем последующих версий
   2.4 Ограничения и исключения
3. Бизнес-контекст
   3.1 Профили заинтересованных лиц
   3.2 Приоритеты проекта
   3.3 Особенности развертывания

Рис. 5-3. Рекомендуемый шаблон документа о концепции и границах

Использование шаблонов

Шаблоны обеспечивают единообразие при организации информации от проекта к проекту. Они позволяют мне помнить то, что я могу забыть, если начну с чистого листа.

Я не заполняю шаблон последовательно сверху вниз, а заполняю разные разделы по мере накопления информации в процессе работы над проектом. Пустые разделы подчеркивают пробелы в текущем знании. Допустим, что один раздел шаблона моего документа называется «Бизнес-риски». В процессе работы над проектом, я обнаруживаю, что раздел пустой. Действительно ли в проекте отсутствуют бизнес-риски? Может мы определили эти риски, но зафиксировали их в другом месте? Или, может, мы еще не работали с соответствующими заинтересованными лицами над определением возможных рисков? Пустые разделы в шаблоне помогают мне более тщательно исследовать важную для проекта информацию. Если вы задаете стандартные вопросы при сборе информации для определенного раздела, стоит указать их в этом разделе шаблона, например в виде скрытого текста, чтобы другие могли использовать их.

При работе с шаблонами я использую термин «усадка для подгонки». Я начинаю с развитого шаблона со многими категориями, которые могут быть важны. После этого я сокращаю его до того, что мне нужно в конкретной ситуации. Допустим, что определенный раздел шаблона, скажем бизнес-риски, не нужен в текущем проекте. Я могу удалить этот раздел из документа или оставить только заголовок, а содержимое убрать. В обоих вариантах есть риск, что читатель заметит пробел и поинтересуется, есть ли какие-нибудь бизнес-риски. Наилучшее решение — разместить в этом разделе явное сообщение: «Бизнес-рисков не обнаружено».

Если какие-то разделы шаблона используются редко, удалите их. Можно создать набор небольших шаблонов для использования в проектах разных типов, например шаблон SRS для работы над большими новыми проектами, для маленьких веб-сайтов и для проектов доработки. Даже если вы храните свои требования в репозитории, а не в традиционном документе, шаблон может помочь вам учесть всю необходимую информацию требований, которую нужно собрать для проекта.

Один менеджер проектов так описал преимущества, полученные от внедрения документа требований: «Их заполнение занимает много времени. Первые несколько раз, когда я их заполнял, я был удивлен объемом подробных сведений, которые нужны, чтобы от документа была польза, а после этого нужен большой объем работы, чтобы привести документ в порядок, избавиться от двусмысленностей, заполнить пробелы и т. п. Но оно того стоило. Первые два продукта, разработанные после внедрения шаблонов, были выпущены вовремя, и качество их было намного выше, чем раньше».

1. Бизнес-требования

Проекты запускаются с полным убеждением, что новый продукт сделает мир для кого-то лучше и обеспечит прибыль. Бизнес-требования описывают основные преимущества, которые новая система даст ее заказчикам, покупателям и пользователям. Бизнес-требования непосредственно влияют на то, какие пользовательские требования будут реализованы и в какой последовательности.

1.1 Исходные данные

Они суммируют обоснование и содержание нового продукта или изменения, которые нужно внести в существующий продукт. Здесь помещают общее описание предыстории или ситуации, в результате чего было принято решение о создании продукта.

1.2 Возможности бизнеса

Для корпоративной информационной системы описывают бизнес-задачу, которая решается посредством этого продукта, или бизнес-процессы, для улучшения которых требуется продукт, а также среду, в которой система будет использоваться. Для коммерческого продукта описывают существующие рыночные возможности и рынок, на котором продукту придется конкурировать с другими продуктами. Этот раздел может содержать сравнительную оценку существующих продуктов и возможных решений, указывая, в чем заключается привлекательность продукта и его преимущества. Опишите задачи, которые не удается разрешить без предлагаемого решения. Покажите, насколько оно соответствует тенденциям рынка, развитию технологий или корпоративной стратегии. Кратко опишите другие технологии, процессы или ресурсы, необходимые для удовлетворения клиента.

Опишите потребности типичных клиентов или целевого рынка. Представьте задачи клиента, которые будет решать новый продукт. Предоставьте примеры того, как клиенты будут использовать продукт. Укажите все известные критичные требования к качеству или интерфейсам, но не упоминайте особенности реализации и дизайна.

1.3 Бизнес-цели

Суммирует важные преимущества бизнеса, предоставляемые продуктом, в количественном и измеряемом виде. Банальности («стать компанией мирового класса») и нечетко сформулированные улучшения («обеспечить высокий уровень сервиса для клиентов») нельзя считать ни полезными, ни поддающимися проверке. В табл. 5-1 приведены примеры и финансовых и нефинансовых целей (Wiegers, 2007).

Табл. 5-1. Примеры финансовых и нефинансовых бизнес-целей

Финансовые цели	Нефинансовые цели
Освоить X% рынка за Y месяцев	Достигнуть показателя удовлетворения покупателей, равного по крайней мере X, в течение Y месяцев со времени выпуска продукта
Увеличить долю рынка в стране W с X% до Y% за Z месяцев	Увеличить производительность обработки транзакций на X% и снизить уровень ошибок данных до величины не более Y%
Достигнуть объема продаж X единиц или дохода в Y долларов за Z месяцев	Разработать надежную платформу для семейства связанных продуктов
Получить X% прибыли по инвестициям в течение Y месяцев	Разработать специальную базовую технологическую основу для организации
Достигнуть положительного потока денежных средств по этому продукту в течение Y месяцев	Добиться признания продукта лучшим по надежности в опубликованных обзорах продуктов к определенной дате
Сэкономить X долларов в год, которые в настоящий момент расходуются на обслуживание унаследованной системы	Обеспечение выполнения определенных нормативов регулирующих органов
Уменьшить затраты на поддержку с X до Y долларов за Z месяцев	Получить не более X звонков в службу обслуживания по каждой единице товара и Y звонков по гарантии каждой единице товара в течение Z месяцев после выпуска продукта
Увеличить валовую маржу для существующего бизнеса с X до Y% в течение одного года	Уменьшить время выполнения заявки до X часов на Y% звонков в службу поддержки

Организации обычно предпринимают проект для решения задачи или использования имеющейся возможности. Модель бизнес-целей отображает иерархию связанных бизнес-проблем и измеряемых бизнес-целей (Beatty и Chen, 2012). Проблемы описывают, что не позволяет организации достичь требуемых ориентиров в настоящее время, тогда как бизнес-цели определяют способы измерения достижения этих ориентиров. Задачи и цели взаимосвязаны: понимание одной раскрывают суть второй.

Имея набор бизнес-целей, задайтесь вопросом: «Что мешает нам достичь этот ориентир?», чтобы определить более подробную бизнес-задачу. Или можно вернуться назад, задав вопрос: «Почему нам вообще важен этот ориентир?», чтобы лучше понять бизнес-задачу или возможность верхнего уровня. При наличии бизнес-задачи спросите себя: «Как определить, что задача решена?», чтобы определить измеряемую цель. Процесс выполняется итеративно путем передвижения по иерархии задач и целей, пока не получится список функций, которые позволят решить задачи для достижения целей.

Разговор между бизнес-аналитиком и куратором, одним из топ-менеджеров компании, с целью определить бизнес-задачи и цели может выглядеть, как показано на рис. 5-4. Это иллюстрация к проекту системы Chemical Tracking System в компании Contoso Pharmaceuticals, описанной в главе 2. На основе ответов топ-менеджера компании бизнес-аналитик может сформулировать бизнес-цели для Chemical Tracking System, как показано на рис. 5-5.

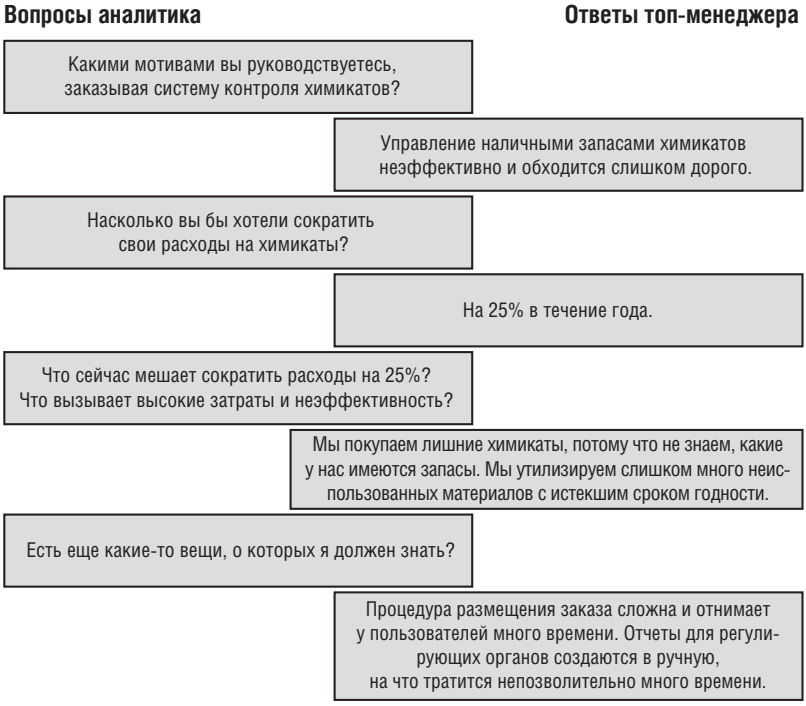


Рис. 5-4. Пример разговора между аналитиком и куратором проекта

1.4 Критерии успеха

Определите, как заинтересованные лица будут определять и измерять успех проекта (Wiegers, 2007). Установите факторы, которые максимально влияют на успех проекта — те, которые организация может контролировать, и те, которые находятся вне сферы ее влияния.

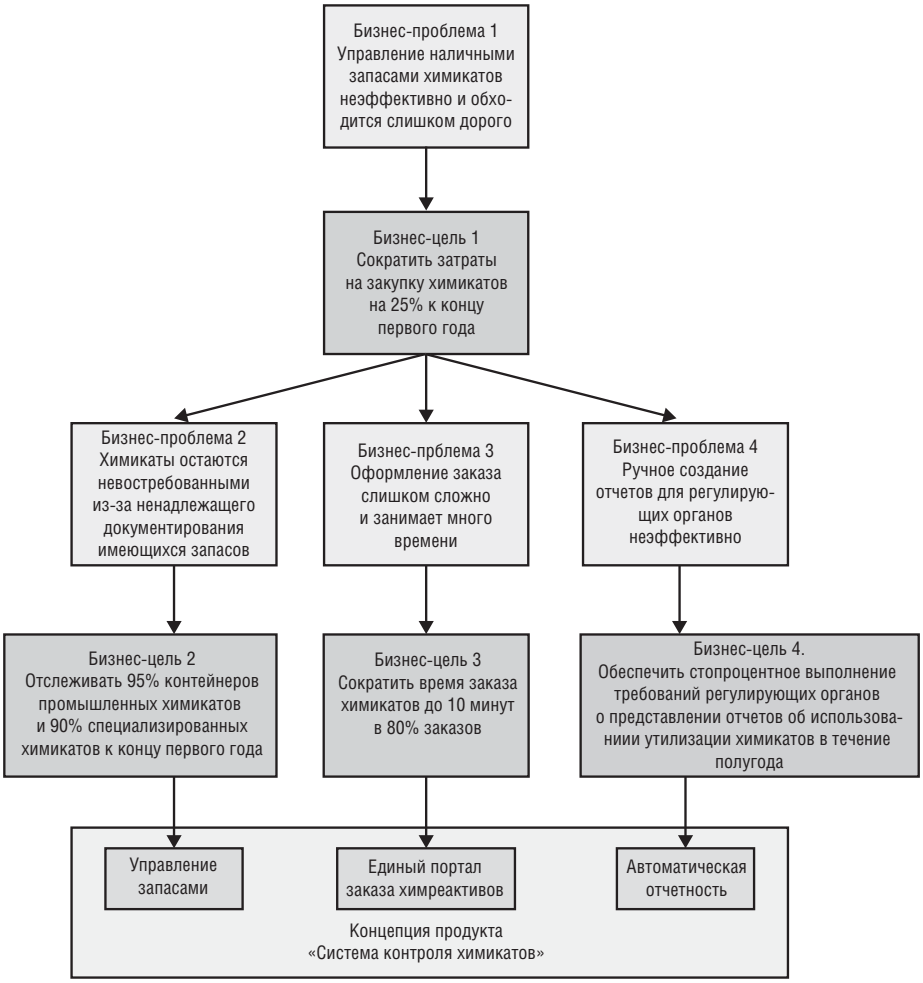


Рис. 5-5. Пример модели бизнес-целей для системы контроля химикатов

Бизнес-цели иногда невозможно измерить, пока не завершиться проект. В других случаях достижение бизнес-целей может зависеть от других проектов. Но все равно важно оценить успех конкретного проекта. Критерии успеха указывают, находится ли проект на пути достижения бизнес-целей. Эти критерии могут определяться во время тестирования или сразу после выпуска продукта. Для системы контроля химикатов одним из критериев может быть Бизнес-цель 3 на рис. 5-5: «Сократить время заказа химикатов до 10 минут в 80% заказов», потому что среднее время заказа можно измерить во время тестирования или после выпуска. Другой критерий успеха может быть связан с Бизнес-целью 2 с временем, намного более ранним, чем год после выпуска, например: «Отслеживать 60% контейнеров промышленных химикатов и 50% специализированных химикатов через четыре месяца».

Внимание! Внимательно выбирайте критерии успеха. Убедитесь, что они оценивают то, что важно для бизнеса, а не просто то, что легко оценить. Критерий «Сократить затраты на производство продукта на 20%» легко оценить. Его также легко реализовать путем увольнения сотрудников или сокращения инвестиций в инновации. Но это могут быть не те результаты, которые подразумевались в целях.

1.5 Положение о концепции

Составьте сжатое положение о концепции проекта, обобщающее долгосрочные цели и назначение нового продукта. В этом документе следует отразить сбалансированную концепцию, удовлетворяющую различных заинтересованных лиц. Она может быть несколько идеалистичной, но должна основываться на существующих или предполагаемых рыночных факторах, архитектуре предприятия, стратегическом направлении развития корпорации или ограничениях ресурсов. Далее показан шаблон, состоящий из ключевых слов, которые прекрасно подходят для документа о концепции продукта (Moore, 1991):

- ***Для*** [целевая аудитория покупателей];
- ***Который*** [положение о потребностях или возможностях];
- ***Эта (этот)*** [имя продукта];
- ***Является*** [категория продукта];
- ***Который(ая)*** [основные функции, ключевое преимущество, основная причина для покупки или использования];
- ***В отличие от*** [основной конкурирующий продукт, текущая система или текущий бизнес-процесс];
- ***Наш продукт*** [положение об основном отличии и преимуществе нового продукта].

Вот как выглядит положение о концепции системы контроля химикатов Chemical Tracking System в Contoso Pharmaceuticals, о которой говорилось в главе 2; ключевые слова выделены полужирным:

> ***Для*** *ученых,* ***которым*** *нужно запрашивать контейнеры с химикатом,* ***данная*** *система Chemical Tracking System* ***является*** *информационной системой,* ***которая*** *обеспечит единую точку доступа к складу химикатов и к поставщикам. Система будет знать местоположение каждого контейнера с химикатом в компании, количество химиката в контейнерах и полную историю перемещения и использования каждого контейнера. Эта система сэкономит компании 25% затрат на химикаты в первый год работы, позволив полностью использовать уже полученные химикаты, утилизировать меньшее количество частично израсходованных или просроченных химикатов и применять единую стандартную систему приобретения химикатов.* ***В отличие*** *от действующих сейчас ручных механизмов заказов химикатов* ***наш продукт*** *будет генерировать все отчеты, необходимые для регулирующих органов, в которых требуются сведения об использовании, хранении и утилизации химикатов.*

Разработка концепции продукта

Как консультант я использую положение о концепции в своей работе. Мы давно работаем с одним клиентом по имени Билл, который время от времени просит нас заняться очередным новым проектом, который обычно имеет свои особенности. Если мы не понимаем, чего именно он хочет, я прошу его написать документ о концепции. Билл всегда немного ворчит, потому что знает, что это заставит его хорошо подумать о том, какой конкретно результат он ожидает получить. Но созданный Биллом документ концепции всегда дает четкое понимание, что нужно сделать, и мы можем эффективно сотрудничать. Это стоит потраченного времени.

Можно попросить несколько заинтересованных лиц написать свое положение раздельно, а не в группе. Сравнение этих положений о концепции будет хорошим способом обнаружить различия в понимании целей проекта. К тому же писать положение о концепции никогда не поздно. Даже если проект уже в работе, разработка положения о концепции обеспечит правильное направление оставшейся части проекта. Положение о концепции создается быстро, но создание правильного положения и согласование его со всеми заинтересованными лицами займет больше времени.

1.6 Бизнес-риски

Обобщает важнейшие бизнес-риски, связанные с разработкой — или не с разработкой — этого продукта. В категорию рисков входят рыночная конкуренция, временны*е* факторы, приемлемость для пользователей, проблемы, связанные с реализацией, и возможные негативные факторы, влияющие на бизнес. Бизнес-риски отличаются от рисков проекта, которые обычно связаны с доступностью ресурсов и особенностями технологий. Оцените возможные потери от каждого фактора риска, вероятность его возникновения, вашу способность контролировать его, а также определите все возможные действия по смягчению ситуации. Подробнее о бизнес-рисках см. главу 32.

1.7 Предположения и зависимости

Предположение (assumption) — это утверждение, которое предполагается верным в отсутствие знаний или доказательств иного. Бизнес-предположения тесно связаны с бизнес-требованиями. Неверные предположения могут не позволить достичь поставленных бизнес-целей. Например, куратор из руководства компании может поставить бизнес-цель увеличить доход на 100 тыс. долларов в месяц. Определяя такую цифру, куратор сделал определенные предположения, например что новый сайт будет привлекать 200 дополнительных уникальных посетителей в день и что каждый посетитель в среднем будет тратить 17 долларов. Если новый сайт не привлечет достаточно посетителей, тратящих достаточное количество денег, проект может не достичь своей бизнес-цели. Если окажется, что те или иные предположения не оправ-

дались, можете изменить границы или график проекта или запустить другие проекты, чтобы достичь другие цели.

Задокументируйте все предположения, сделанные заинтересованными лицами, когда они обдумывали проект и создавали данный документ о концепции и границах. Часто предположения одних лиц не разделяют другие стороны. Если вы запишите их и просмотрите позже, то избежите возможной путаницы и ухудшения ситуации в будущем.

Задокументируйте важнейшие зависимости проекта от внешних факторов — изменения отраслевых стандартов или предписаний регулирующих органов, других проектов, сторонних поставщиков или партнеров по разработке. Предположения и зависимости бизнеса могут превратиться в риски, которые должен регулярно отслеживать менеджер проекта. Нарушение зависимостей — популярная причина задержек проекта. Опишите возможные последствия того, что предположения окажутся ошибочными или зависимости нарушены, чтобы заинтересованные лица могли понять, почему это так важно.

2. Рамки и ограничения проекта

Когда химик изобретает новую химическую реакцию, которая преобразует одно вещество в другое, он пишет документ, в который входит раздел «Рамки и ограничения», где описывает, что получится и не получится в результате этой реакции. Точно так же для проекта по разработке ПО следует определить его рамки и ограничения. Вам необходимо указать, что *может* делать система, а что *не может*.

Многие проекты страдают «распуханием границ» — резким ростом из-за активного расширения функциональности продукта. Первый шаг на пути к обузданию распухания границ — определение рамок проекта. Границы проекта определяют концепцию и круг действия предложенного решения. В ограничениях указываются определенные возможности, которые *не будут* включены в продукт. Рамки и ограничения помогают установить реалистичные ожидания заинтересованных лиц, потому что иногда клиенты запрашивают функции, слишком дорогостоящие или выходящие за предполагаемые границы продукта.

Рамки проекта могут представляться различными способами (см. раздел «Способы представления границ проекта» в этой главе). На самом высоком уровне границы определяются, когда клиент решает, какие бизнес-цели преследовать. На низком уровне границы определяются на уровне функций, пользовательских историй, вариантов использования или событий и реакции на них. В конечном итоге границы определяются через набор функциональных требований, которые планируется реализовать в определенном выпуске или итерации. На каждом уровне границы не должны выходить за рамки более высокого уровня. Например, находящиеся в границах пользовательские требования должны соответствовать бизнес-целям, а функциональные требования — пользовательским требованиям в заданных границах.

Нереалистичные требования

Однажды менеджер в компании-разработчике, проект которого страдал от практически катастрофического распухания границ, печально сказал мне: «Мы слишком нереалистично отнеслись к требованиям». Он имел в виду, что все идеи, какие у кого-либо были, включили в требования. У компании крепкая концепция продукта, но в ней не управляли границами проекта путем планирования серии выпусков и откладывания некоторых функций на более поздние выпуски (или навсегда). В конце концов, команда выпустила «перекачанный» продукт лишь после четырех лет разработки. Может оказаться ценным учесть нереалистичные требования в будущих выпусках, но вдумчивое управление границами проекта и поэтапный подход к разработке позволил бы команде выпустить продукт намного раньше.

2.1 Основные функции

Опишите основные функции продукта или возможности пользователей, уделив основное внимание тому, что отличает продукт от предыдущей версии или конкурирующих продуктов. Подумайте, как пользователи будут работать с этими функциями, чтобы убедиться, что список функций полон и не содержит ненужных функций, которые интересны, но не приносят пользы пользователям. Назначьте каждой функции уникальное и постоянное название, чтобы ее можно было отслеживать в других компонентах системы. Можно создать древовидную схему, как показано далее в этой главе.

2.2 Объем первоначально запланированной версии

Обобщает основные запланированные функции, включенные в первоначальную версию продукта. Границы проекта обычно определяются как набор функций, но их можно также определять в терминах пользовательских историй, вариантов использования, потоков вариантов использования или внешних событий. Также можно описать характеристики качества, которые позволят продукту предоставлять предполагаемые преимущества различным классам пользователей. Если ваша задача — сосредоточиться на разработке и уложиться в график, вам следует избегать искушения включить в версию 1.0 каждую функцию, которая когда-нибудь в будущем может понадобиться какому-то потенциальному покупателю. Распухание кода и сдвиг графика — типичные исходы такого коварного набивания объема. Сосредоточьтесь на наиболее ценных функциях, имеющих максимально приемлемую стоимость, годных для самой широкой целевой аудитории, которые удастся создать как можно раньше.

В качестве иллюстрации приведу недавний проект, в котором команда решила, что пользователи должны иметь возможность запускать собственную службу доставки в первой версии ПО. Версия 1 не обязательно должна быть быстрой, красиво оформленной или легкой в использовании, но она должна быть надежной; этим принципам команда следовала четко. Первая версия си-

стемы выполняет лишь основные задачи. В будущие выпуски будут включены дополнительные функции, возможности и средства, обеспечивающие легкость и простоту использования. Но в первом выпуске важно не забыть о нефункциональных требованиях. Тех, что непосредственно влияют на архитектуру, их особенно важно установить с самого начала. Переделка архитектуры для исправления недостатков качества может стоить столько же, сколько написание продукта с нуля. Подробнее об атрибутах качества ПО см. главу 14.

2.3 Объем последующих версий

Если вы ожидаете поэтапную эволюцию продукта или если используете итеративную модель разработки, создайте план выпуска, в котором укажите, какие функции будут отложены и желательные сроки последующих выпусков. В последующих версиях вы сможете реализовать дополнительные варианты использования и функции и расширить возможности первоначальных вариантов использования и функций. Чем дальше вы заглядываете, тем более расплывчатыми будут границы проекта. Вам наверняка придется передвинуть функциональность с одного запланированного выпуска до другого и, возможно, добавлять незапланированные функции. Короткие циклы выпусков часто предоставляют удобные случаи для накопления знаний, основанных на отзывах клиентов.

2.4 Ограничения и исключения

Перечислите все возможности или характеристики, которых могут ожидать заинтересованные в проекте лица, но включение которых в продукт или в определенную версию не запланировано. Перечислите изъятые элементы, чтобы не забыть решения по границам проекта. Если пользователь запросил возможность доступа к системе с телефона, когда он не находится на рабочем месте, и эта функция была признанной не входящей в границы проекта, тогда четко запишите в соответствующем разделе: «Новая система не поддерживает доступа с мобильных устройств».

3. Бизнес-контекст

В этом разделе представлены профили основных категорий заинтересованных лиц, приоритеты руководства в проекте, а также сводка некоторых обстоятельств, которые надо учесть при планировании развертывания решения.

3.1 Профили заинтересованных лиц

Заинтересованными в проекте лицами (stakeholders) называются отдельные лица, группы или организации, которые активно вовлечены в проект, на которых влияет результат проекта и которые сами могут влиять на этот результат (Smith, 2000; IIBA, 2009; PMI, 2013). Профили заинтересованных лиц описывают различные категории клиентов и других ключевых лиц, заинтересованных в этом проекте. Вам не нужно описывать каждую группу заинтере-

сованных лиц, например юристов, которые должны проверять соответствие надлежащим законам. Сферой вашего интереса должны стать различные группы клиентов, целевые рыночные сегменты и различные классы пользователей, входящих в эти сегменты. В профиль каждого заинтересованного в проекте лица включается следующая информация:

- основная ценность или преимущество, которое продукт принесет заинтересованным лицам, и то, как продукт удовлетворит покупателей. Ценность для заинтересованных лиц представляют:
  - повышенная производительность;
  - меньшее количество переделок;
  - снижение себестоимости;
  - ускорение бизнес-процессов;
  - автоматизация задач, ранее выполнявшихся вручную;
  - возможность выполнять совершенно новые задачи;
  - соответствие соответствующим стандартам и правилам;
  - лучшая, по сравнению с текущими продуктами, легкость и простота использования;
- их вероятное отношение к продукту;
- самые важные для них функции и характеристики;
- все известные ограничения, которые должны быть соблюдены.

Можно включить поименный список ключевых заинтересованных лиц для каждого профиля или структурную схему организации, показывающую отношения между заинтересованными лицами в организации.

3.2 Приоритеты проекта

Чтобы принимать эффективные решения, заинтересованные лица должны договориться о приоритетах проекта. Один из подходов к этому заключается в рассмотрении пяти измерений: функции (или объем), качество, график, затраты и кадры (Wiegers, 1996a). В любом проекте каждое из этих измерений относится к одной из трех категорий:

- ***ограничение*** — сдерживающий фактор, в рамках которого должен оперировать менеджер проекта;
- ***ведущий фактор*** — важный фактор успеха, ограниченно гибкий при изменениях;
- ***степень свободы*** — возможность для менеджера проекта до определенной степени менять измерение и балансировать относительно других измерений.

Задача менеджера проекта — скорректировать те факторы, которые представляют собой степени свободы для достижения ключевых факторов успеха проекта в рамках, налагаемых ограничениями. Представьте себе, что отдел маркетинга неожиданно требует создать продукт на месяц раньше срока. Какова будет ваша реакция? Возможные варианты ответов:

- Вы отложите реализацию определенных требований до более поздней версии.
- Сократите запланированный цикл тестирования системы.
- Оплатите сверхурочную работу ваших специалистов или пригласите специалистов по контракту для ускорения разработки.
- Привлечете ресурсы других проектов для разрешения ситуации.

Ваши действия в подобных ситуациях зависят от приоритетов проекта. В реальности при возникновении изменения вам нужно поговорить с ключевыми заинтересованными лицами, чтобы определить ответные действия. Например, отдел маркетинга может потребовать добавить новые функции или сократить длительность проекта, возможно в обмен на отказ от реализации некоторых функций. В Приложении В вы найдете пример, как нужно документировать приоритеты проекта.

> **Внимание!** Не все пять измерений могут быть ограничениями, и они не могут быть ведущими факторами. Менеджеру проекта нужна определенная степень свободы, чтобы иметь возможность реагировать при изменении требований или обстоятельств проекта.

3.3 Особенности развертывания

Перечислите информацию и действия, необходимые для обеспечения эффективного развертывания решения в рабочую среду. Опишите доступ, который потребуется пользователями для работы с системой, в частности, находятся ли они далеко в разных часовых поясах или недалеко друг от друга. Укажите, когда пользователям в разных местах нужен доступ к системе. Если требуются изменения инфраструктуры, чтобы обеспечить потребности ПО в мощностях, доступе к сети, хранилищу данных и миграции данных, опишите эти изменения. Зафиксируйте всю информацию, которая потребуется тем, кто будет готовить бизнес-процессы обучения и модификации в связи с развертыванием нового решения.

Способы представления границ проекта

Описанные в этом разделе модели могут использоваться для различного представления границ проекта. Не нужно создавать все эти модели — выберите одну, которая даст самую информативную картину проекта. Модели можно включать в документ концепции и границ или хранить в другом месте для использования в дальнейшем.

Задача таких инструментов, как контекстная диаграмма, карта экосистемы, дерево функций и список событий, — поощрять прозрачные и точные механизмы общения между заинтересованными лицами проекта. Такая прозрачность важнее догматичного соблюдения всех правил для создания «правильной» диаграммы. Однако мы настоятельно рекомендуем придерживаться применяемой в следующих примерах нотации как стандарту соз-

дания диаграмм. Например, если в контекстной диаграмме вы используете для представления системы треугольники вместо круга, а для представления внешних сущностей — овалы вместо прямоугольников, вашим коллегам будет тяжело читать такую «нестандартную» диаграмму.

Для наглядного представления границ проекта чаще всего используют контекстные диаграммы, карты экосистемы, дерева функций и списки событий. Но применяются и другие методы. Выявление затрагиваемых бизнес-процессов также помогает определять границы проекта. Диаграммы вариантов использования позволяют проиллюстрировать границу между вариантами использования и действующими лицами (см. главу 8).

Контекстная диаграмма

Уточнение рамок определяет границу и связи разрабатываемой системы со всем остальным миром. *Контекстная диаграмма* (context diagram) графически иллюстрирует эту границу. Она определяет *оконечные элементы* (terminators), расположенные вне системы, которые определенным образом взаимодействуют с ней, а также данные, элементы управления и материальные *потоки*, протекающие между оконечными элементами и системой. Контекстная диаграмма представляет собой высший уровень абстракции в диаграмме потока данных, разработанной по принципам структурного анализа (Robertson и Robertson, 1994), но эта модель полезна и в других проектах.

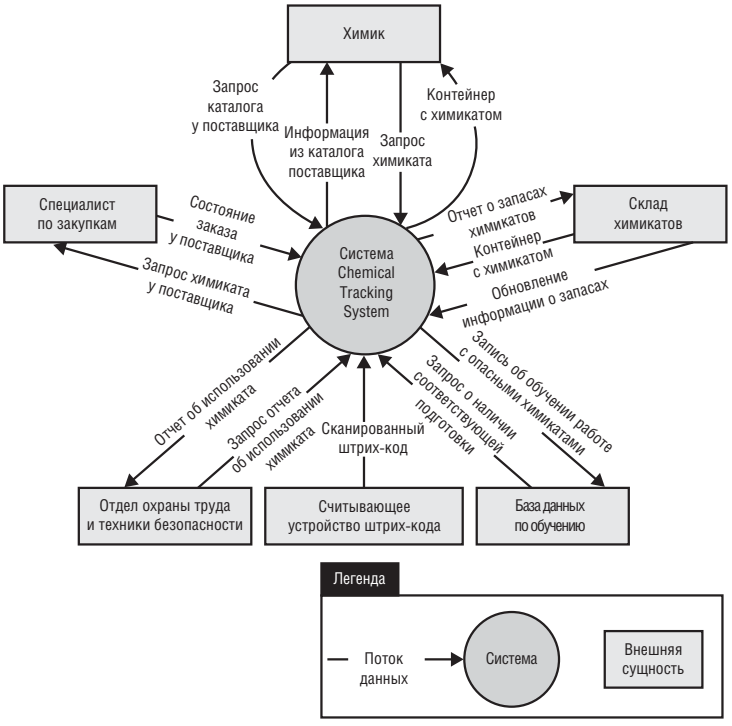


Рис. 5-6. Частичная контекстная диаграмма для системы Chemical Tracking System

На рис. 5-6 показана часть контекстной диаграммы для Chemical Tracking System. Вся система изображена кружком; на контекстной диаграмме намеренно не показывают внутренние объекты системы, процессы и данные. «Система» внутри кружка может иметь любую комбинацию ПО, оборудования или человеческих ресурсов. Поэтому она может содержать ручные операции в составе системы в целом. Оконечные элементы в прямоугольниках представляют классы пользователей («Химик» или «Специалист по закупкам»), отделы («Отдел охраны труда и техники безопасности»), другие системы («База данных по обучению») или аппаратные устройства («Считывающее устройство штрих-кода»). Стрелками показаны потоки данных («запрос химиката») или физические элементы («контейнер с химикатом») между системой и оконечными элементами.

Можно было бы ожидать, что поставщики химикатов будут показаны на диаграмме в виде оконечных элементов. Ведь компания направляет заказы для выполнения поставщикам, а те отправляют контейнеры с химикатами и счета в Contoso Pharmaceuticals, отдел же закупок платит поставщикам. Однако эти процессы происходят вне Chemical Tracking System, как часть операций отделов закупок и приобретений. Их отсутствие в контекстной диаграмме показывает, что система не участвует напрямую в размещении заказов у поставщиков, в получении товаров или оплате счетов.

Карта экосистемы

Карта экосистемы (ecosystem map) показывает все системы, связанные с создаваемой системой и взаимодействующие друг с другом, а также природу этих взаимодействий (Beatty и Chen, 2012). Карта экосистемы представляет рамки путем отображения всех систем, которые взаимосвязаны друг с другом и которые может потребоваться изменить при создании вашей системы. Карта экосистемы отличается от контекстных диаграмм тем, что показывает другие системы, связанные с создаваемой вами, в том числе без непосредственных интерфейсов. Зависимые системы можно определить путем выявления тех, что потребляют данные, поступающие из вашей системы. Когда вы достигнете точки, в которой ваш проект не влияет ни на какие дополнительные данные, можно сказать, что вы достигли границы систем, которые относятся к решению.

На рис. 5-7 показана частичная карта экосистемы для Chemical Tracking System. Системы изображены в виде прямоугольников (например, «Система закупок» или «Приемная система»). В этом примере основная система, над которой мы работаем, выделена жирным прямоугольником («Chemical Tracking System»), но если у всех систем равный статус, можно применить такой стиль и к ним. Линии обозначают интерфейсы между системами (например, от системы закупок к Chemical Tracking System). Линии со стрелками и надписями показывают важные порции данных, переходящих от одной системы к другой (например, «Записи об обучении работе с опасными

химикатами» передаются от «Корпоративная база данных по обучению» в «Chemical Tracking System»). Некоторых из этих потоков также могут присутствовать на контекстной диаграмме.

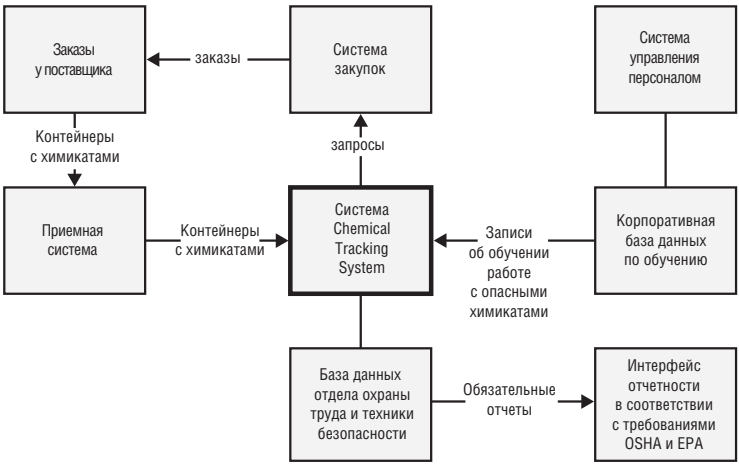


Рис. 5-7. Частичная карта экосистемы для Chemical Tracking System

Карта экосистемы на рис. 5-7 показывает, что Chemical Tracking System не подключается напрямую к интерфейсу поддержки отчетов в соответствии с Законом о гигиене и безопасности труда на рабочем месте (OSHA) и требованиями Агентства по охране окружающей среды (EPA). Тем не менее, нужно учитывать, что в Chemical Tracking System могут возникнуть требования, связанные с данными переходящими от нее на интерфейс отчетности через базу данных отдела охраны труда и техники безопасности.

Дерево функций

Дерево функций (feature tree) представляет собой наглядную картину функций, объединенных в логические группы с иерархическим разбиением каждой функций на более мелкие (Beatty и Chen, 2012). Дерево функций предоставляет сжатую иллюстрацию всех запланированных к реализации в проекте функций, что отлично подходит для показа топ-менеджерам, желающим увидеть общую картину всего проекта. Дерево функций может содержать до трех уровней функций, которые обычно называют уровень 1 (L1), уровень 2 (L2) и уровень 3 (L3). Функции уровня L2 являются подфункциями L1, а функции L3 — подфункциями L2.

На рис. 5-8 показана частичная карта экосистемы для Chemical Tracking System. Ствол дерева в середине представляет реализуемый продукт. У каждой функции собственная линия или «ветка», отходящая от ствола. Серые прямоугольники представляют функции уровня L1, такие как «Приобретение химикатов» и «Управление запасами». Линии, отходящие от L1, представляют функции уровня L2: «Поиск» и «Заказ химикатов» являются подфункциями функции «Приобретение химикатов». Подветками ветки L2 являются

функции уровня L3: «Поиск в локальных лабораториях» является подфункцией функции «Поиск».

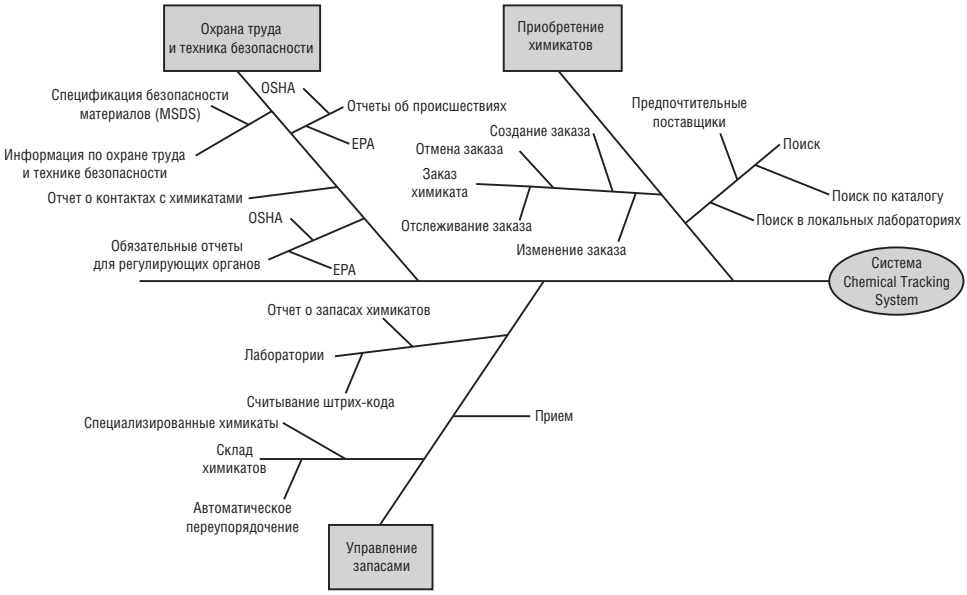


Рис. 5-8. Частичная карта функций для системы Chemical Tracking System

При планировании выпуска или итерации их границы можно определить путем выбора для реализации определенного набора функций и подфункций (Nejmeh и Thomas, 2002; Wiegers, 2006). Функцию можно было реализовать целиком в определенном выпуске, или можно реализовать только ее часть, выбирая определенные подфункции из уровней L2 и L3. В последующих выпусках можно дополнить эти рудиментарные реализации путем добавления большего числа подфункций уровней L2 и L3, пока все функции не будут реализованы в конечном продукте. Так что границы определенного выпуска состоят из определенного набора функций уровней L1, L2 и/или L3 из дерева функций. Выбор функций для реализации в разных выпусках можно отметить цветами или шрифтом. С другой стороны можно создать таблицу с планом подфункций реализуемых в каждом выпуске (Wiegers, 2006).

Список событий

Список событий (event list) перечисляет внешние события, которые могут инициировать определенное поведение в системе. Список событий определяет границы системы путем перечисления возможных бизнес-событий, инициируемых пользователями или инициируемых временем (срабатывание по времени), или сигналов от внешних компонентов, таких как аппаратные устройства. В списке находятся только названия событий — функциональные требования, описывающие, как система реагирует на события, должны описываться в спецификации SRS с использование таблиц событий и реак-

ций на них. Подробнее о таблицах событий и реакций см. главу 12.

На рис. 5-9 показан частичный список событий для Chemical Tracking System. В каждом элементе списка указывается, что инициирует событие («Химик» делает что-то или наступает «Время запуска»), а также действие по событию. Список событий также хорошее средство разграничения, потому что можно назначать реализацию определенных событий в конкретном выпуске продукта или итерации разработки.

Внешние события для Chemical Tracking System

- Химик разместил заказ химиката.
- Просканирован штрих-код контейнера с химикатом.
- Наступило время генерации отчетов OSHA.
- Поставщик выпустил новый каталог химикатов.
- Новый специализированный химикат добавлен в систему.
- Поставщик отменил заказ химиката.
- Химик запросил свой отчет о контактах с химикатами.
- Получена спецификация безопасности материалов из Управления по охране окружающей среды (EPA).
- В список предпочтительных поставщиков добавлен новый поставщик.
- Получен контейнер с химикатами от поставщика.

Рис. 5-9. Частичный список событий для Chemical Tracking System

Обратите внимание, как список событий дополняет контекстную диаграмму и карту экосистемы. Контекстная диаграмма и карта экосистемы в совокупности описывают внешних действующих лиц и задействованные системы, а список событий определяет, как эти действующие лица и системы могут вызвать определенное поведение в создаваемой системе. Список событий можно сверить на предмет корректности и полноты с контекстной диаграммой и картой экосистемы следующим образом:

- Определите, какие внешние сущности в контекстной диаграмме могут являться источниками событий: «Могут ли какие-либо действия химика инициировать определенное поведение системы Chemical Tracking System?»
- Посмотрите, нет ли в карте экосистемы системы, которая может инициировать события в вашей системе.
- Для каждого события определите, если соответствующие ему внешние сущности в контекстной диаграмме или системы в карте экосистемы: «Если контейнер с химикатом может поступить от поставщика, может ли поставщик фигурировать в контекстной диаграмме и/или в карте экосистемы?»

Обнаружив несоответствие, посмотрите внимательнее — может в модели отсутствует какой-то элемент. В данном случае в контекстной диаграмме поставщик отсутствует, потому что система Chemical Tracking System не взаимодействует напрямую с поставщиками. Вместе с тем поставщик присутствует в карте экосистемы.

Не упускайте границы из вида

Определение границ — структура, а не смирительная рубашка. Бизнес-требования и понимание того, как клиенты будут использовать продукт, ценны

при работе с границами проекта. В изменении объема как такового нет ничего плохого, если это помогает вам направить проект в сторону удовлетворения растущих потребностей клиентов. Документ о концепции и границах позволяет оценить, действительно ли предложенные функции и требования стоит включать в проект. Можно менять границы будущей итерации или целого проекта, если это делается осознанно, «правильными» людьми, по правильными бизнес-основаниям и с пониманием и принятием всех последствий.

Помните, каждый раз, когда кто-то выдвигает новое требование, аналитик должен спросить: «Попадает ли оно в рамки проекта?» Один из возможных ответов: требование явно выходит за границы проекта. Оно может быть интересным, но его следует переадресовать будущим версиям или другому проекту. Другой ответ: требование попадает в границы проекта. Вы можете включить такие требования в проект, если они обладают высоким приоритетом по сравнению с теми требованиями, которые уже описаны в проекте. При этом зачастую приходится решать, отложить или отклонить другие запланированные требования, если только вы не готовы расширить временные рамки проекта.

Третий вариант ответа: предложенное новое требование выходит за рамки проекта, но идея так хороша, что следует изменить границы проекта, чтобы требование попало в него с соответствующими коррективами бюджета, графика и ресурсов. То есть существует обратная связь между пользовательскими требованиями и бизнес-требованиями. При этом вам придется обновить документ о концепции и границах, изменения в котором должны отслеживаться в системе управления версиями с момента создания его базовой версии. Отслеживайте, почему отвергаются требования, потому что они имеют свойство возникать вновь. В главе 27 рассказывается, как задействовать атрибут требования для отслеживания отвергнутых или отложенных требований.

Использование бизнес-целей для принятия решений о границах проекта

Бизнес-цели — самый важный фактор, который нужно учитывать, принимая решения о границах проекта. Определите, какие из предложенных функций или пользовательских требований приносят самую большую выгоду с точки зрения бизнес-целей и запланируйте их на самые первые выпуски. Когда заинтересованное лицо просит добавить функциональность, определите, как это изменение соотносится с бизнес-целями. Например, бизнес-цель, предусматривающая получение максимального дохода от интерактивного терминала, подразумевает раннюю реализацию функций, которые способствуют продаже бо́льшего объема товаров и услуг клиентам. Не стоит назначать высокий приоритет броским функциям, которые интересуют только любителей новых технологий и ничего не привносят в достижение бизнес-цели.

Если возможно, оцените количественно вклад функции в достижение бизнес-цели, чтобы решения о границах принимались на основе фактов, а не эмоций (Beatty и Chen, 2012). Какой будет вклад новой функции — тысяча,

сто тысяч или миллион? Когда топ-менеджер просит добавить новую функцию, которую придумал за выходные, нужно применить численный анализ, чтобы определить, правильно ли бизнес-решение добавить ее в продукт.

Оценка эффекта от изменения границ проекта

При увеличении границ проекта менеджеру обычно приходится повторно согласовывать запланированный бюджет, ресурсы, график и/или персонал. В идеале, исходный график и ресурсы рассчитаны на возможность определенных изменений благодаря наличию предусмотрительно запланированных резервов на случай непредвиденностей (Wiegers, 2007). В противном случае после одобрения изменений потребуется повторно выполнить планирование.

Обычный результат изменения границ заключается в том, что завершенные действия приходится переделывать. Если при добавлении новой функциональности не увеличиваются выделенные на проект время или ресурсы, часто страдает качество. Задокументированные бизнес-требования облегчают управление изменениями границ проекта при изменении рыночных условий или бизнес-потребностей. Они также облегчают задачу утомленному менеджеру проекта, когда ему нужно сказать «нет» или, по крайней мере, «не сейчас» влиятельным людям, старающимся впихнуть побольше функций в и так напряженный проект.

Концепция и границы в проектах гибкой разработки

Для управления границами проектом гибкой разработки (agile), в котором разработка ведется сериями коротких итераций, нужно применять другой подход. Границы каждой итерации состоят их пользовательских историй, выбранных из динамического резерва (backlog) на основе их относительных приоритетов и расчетной производительности команды в каждой итерации. Вместо того чтобы бороться с распуханием границ, в команде определяют приоритеты новых требований на основе существующих элементов резерва проекта и назначают их на будущие итерации. Число итераций, а значит, и общая продолжительность проекта, все равно определяется общим объемом функциональности, которую нужно реализовать, но границы каждой итерации жестко контролируются, чтобы гарантировать ее своевременное завершение. С другой стороны, в некоторых проектах гибкой разработки фиксируется общая продолжительность проекта, а меняются границы. Число итераций может оставаться фиксированным, но рамки остающихся итераций изменяются в соответствии с относительными приоритетами существующих или новых пользовательских историй.

Команда может определить высокоуровневую дорожную карту итераций вначале проекта, но распределение пользовательских историй по итерациям может выполняться вначале каждой итерации. Обращение к бизнес-

требованиям при определении границ каждой последующей итерации помогает гарантировать, что будет поставлен продукт, отвечающий бизнес-целям. Подобная стратегия может применяться в любом проекте, состоящем из итераций (см. врезку «Управление объемом и разработка по итерациям»).

Управление объемом и разработка по итерациям

Энрике, менеджер проекта в Lightspeed Financial Systems, занимался выпуском интернет-версии ведущего программного продукта Lightspeed для управления портфелем ценных бумаг. Потребовались бы два года для поставки полнофункциональной версии приложения, однако компания должна была дать знать о себе в Интернете уже сейчас. Энрике выбрал прием «разработка по итерациям» и обещал выпускать новую версию каждые 90 дней. Его команда специалистов по маркетингу расставила приоритеты требований к продукту. В спецификацию требований для каждой ежеквартальной версии входил утвержденный набор новых и улучшенных функций, а также список «переходящих» требований низкого приоритета, которые предполагалось реализовать, если позволяло время. Команда Энрике не включала каждое такое требование в каждую версию, но они поставляли новую, стабильную версию каждые три месяца с помощью подхода, который учитывал график и управление границами итерации. График и качество — это естественные ограничения проекта «разработка по графику», а границы итерации представляли собой степень свободы.

Хотя в проектах гибкой разработки обычно и не создается формальный документ концепции и границ, содержимое шаблона на рис. 5-3 и относимо, и обязательно для поставки успешного продукта. Во многих проектах гибкой разработки выполняется начальная (нулевая) итерация планирования для определения общей концепции продукта и других требований проекта.

Бизнес-требования должны определяться во всех проектах разработки ПО независимо от модели разработки. Бизнес-цели описывают ожидаемую выгоду от проекта, а в проекте гибкой разработки они используются для облегчения определения приоритетов резерва (backlog), чтобы добиться максимальной ценности для бизнеса в самых ранних итерациях. Критерии успеха надо определить так, чтобы после развертывания можно было измерить успех и соответствующим образом скорректировать резерв проекта. Положение о концепции содержит долгосрочный план — то, чем должен стать продукт по завершении всех итераций.

Применение бизнес-целей для определения момента завершения проекта

Как узнать, когда можно завершить реализацию функциональности? Обычно управлением проектом и точкой его завершения занимается менеджер. Но бизнес-аналитик близко знаком с бизнес-целями и может помочь определить, когда она достигнута.

Если работа над проектом начинается с четкой концепции и если границы каждого выпуска или итерации включают лишь часть общей функциональности, тогда работу можно считать выполненной по завершении запланированных итераций. В результате завершения всех итераций должен получиться полностью реализованный продукт, удовлетворяющий бизнес-целям.

Однако именно в проектах с итеративной разработкой финальная точка может быть неопределенной. Для каждой итерации определяются ее границы. По мере продвижения проекта резерв незавершенной работы сокращается. Не всегда необходимо реализовать полный набор остающейся функциональности. Критически важно иметь четкие бизнес-цели, чтобы можно было двигаться к их удовлетворению поэтапно, по мере получения информации. Проект готов, когда критерии успеха указывают, что у вас хорошие шансы удовлетворить бизнес-цели. Нечеткие бизнес-цели — гарантия свободного проекта, в котором невозможно определить момент завершения. Финансирующим проект заказчикам это не нравится, потому что непонятно, как в таких проектах определить бюджет, график или план. Клиентам это не нравится, потому что они могут получить решение, которое поставляется в рамках бюджета и графика, но не предоставляет нужных им возможностей. Но этот риск присущ работе над продуктами, которые нельзя точно определить с самого начала, если только вы не уточните бизнес-цели в процессе работы над проектом.

Сосредоточьтесь на определении четких бизнес-требований во всех своих проектах, в противном случае вы будете двигаться без цели, надеясь сделать что-то полезное, но без возможности узнать, в правильном ли направлении вы движетесь.

Что дальше?

- Попросите нескольких заинтересованных в проекте лиц написать положение о концепции, используя шаблон ключевых слов, о котором говорилось в этой главе. Посмотрите, насколько совпадают их представления. Разрешите все нестыковки и составьте объединенное положение, согласовав его со всеми заинтересованными лицами.
- Независимо от того, скоро ли выпуск нового проекта или вы еще в середине процесса, напишите документ о концепции и границах, используя шаблон на рис. 5-3, или просто создайте модель бизнес-целей и попросите остальных членов команды просмотреть его. Может выясниться, что у членов команды нет единого представления о концепции или границах продукта. Устраните эту проблему сейчас; в будущем исправить ее гораздо труднее. Так вы определите несколько вариантов изменения шаблона, что позволит наилучшим образом осуществить различные проекты, выполняемые вашей организацией.
- Запишите измеримые бизнес-цели в своем проекте в формате, который позволяет легко делиться информацией на совещаниях на протяжении проекта. Принесите документ на следующее совещание по требованиям и посмотрите, посчитают ли его другие участники полезным.

Глава 6

Как отобрать пользователей для работы над проектом

Джереми зашел в кабинет Рут Гилберт, директора отдела поиска новых лекарств в компании Contoso Pharmaceuticals. Рут попросила ИТ-команду, обслуживающую исследовательскую организацию Contoso, создать новое приложение, которое позволило бы химикам-исследователям ускорить поиск новых лекарств. Джереми был назначен бизнес-аналитиком проекта. Представившись и обсудив проект в целом, Джереми сообщил Рут: «Я хотел бы поговорить с одним из ваших химиков, чтобы понять их требования к системе. У вас есть хороший специалист, с которым можно было бы поговорить?»

Рут ответила: «Я занималась этой работой пять лет, до того как стала директором отделения три года тому назад. На самом деле вам не нужно разговаривать ни с кем из моих подчиненных — я расскажу вам все, что нужно об этом проекте».

Джереми задумался. Научные знания и технологии меняются быстро, поэтому он не был уверен, что Рут может адекватно представлять текущие и будущие потребности пользователей такого сложного приложения. Возможно имела место какая-то внутрикорпоративная политика, которая заставляла Рут создавать буфер между Джереми и реальными пользователями. Но после небольшого обсуждения стало ясно, что Рут не хочет, чтобы кто-то из ее людей принимал непосредственное участие в проекте.

«Хорошо, нехотя согласился Джереми. — Наверное, я начну с некоторого анализа документов и приду с имеющимися вопросами к вам. Мы можем запланировать ряд интервью на следующие несколько недель, чтобы я мог понять, что ваши ученые ожидают получить от новой системы?»

«Я приношу свои извинения, но сейчас я сильно занята, — ответила Рут. — Я могу посвятить вам несколько часов примерно через три недели на прояснение непонятных вам вещей. Просто приступайте к написанию требований. При встрече вы сможете задать любые вопросы, которые у вас останутся к тому времени. Я надеюсь, что этого вам достаточно, чтобы приступить к работе над проектом».

Если вы разделяете мое убеждение, что участие клиента крайне важно для создания продукта отличного качества, вы должны обеспечить, чтобы бизнес-аналитик и менеджер проекта с самого начала активно работали с нужными представителями клиента. Качество собранных требований к ПО, а следовательно, и успех самого ПО зависит от того, насколько голос пользователя будет услышан разработчиками. Прежде всего необходимо понять, чего хочет пользователь:

- определите различные классы пользователей вашего продукта;
- выберите представителей каждого класса пользователей и прочих групп заинтересованных лиц и поработайте с ними;
- согласуйте с ними, кто будет отвечать за принятие решений по проекту.

Вовлечение клиентов в работу над проектом — единственный способ избежать разрыва ожиданий, описанного в главе 2, — их претензий и разочарования, когда они получают не тот продукт, который ожидали. Недостаточно раз или два расспросить нескольких клиентов или их менеджера о том, как они представляют продукт, чтобы немедля браться за дело. Если разработчики будут выполнять все требования клиентов, им скорее всего только и придется, что переделывать продукт, поскольку клиенты зачастую сами не знают, чего хотят на самом деле. Кроме того, бизнес-аналитики могут разговаривать не с теми людьми или задавать не те вопросы.

Желания пользователей не всегда совпадают с функциональностью, необходимой для выполнения задач с помощью конкретного продукта. Чтобы получить более точное представление о потребностях пользователей, бизнес-аналитик должен опросить клиентов, проанализировать информацию, внести уточнения и определить, что именно нужно пользователям для выполнения работы. На аналитика возложена вся полнота ответственности за документирование функций и свойств системы и за ознакомление с этой информацией всех заинтересованных лиц. Это постепенный процесс, требующий времени. Если вы не готовы потратить время на выработку согласованного представления о желаемом продукте, готовьтесь к неизбежным переделкам, срывам графика и недовольству клиентов.

Классы пользователей

Люди часто говоря о «пользователях» программной системы, как будто все пользователи принадлежат к монолитной группе с похожими характеристиками и потребностями. На самом деле в большинстве продуктов задействованы разнообразные пользователи с различными ожиданиями и целями. Вместо того чтобы мыслить о «пользователе» в единственном числе, потратьте какое-то время на определение разных классов пользователей и их ролей и привилегий по отношению к вашему продукту.

Классификация пользователей

В главе 2 приведено много типов заинтересованных лиц, которые могут быть в проекте. Как показано на рис. 6-1, класс пользователей является подмножеством пользователей продукта, которые являются подмножеством клиентов продукта, а те в свою очередь являются подмножеством заинтересованных лиц. Один человек может принадлежать разным классам пользователей. Так, администратор иногда работает с системой как рядовой пользователь. Помимо всего прочего, пользователей продукта можно подразделять по таким признакам, и вы можете группировать пользователей на отдельные классы на основе следующих различий:

- по привилегиям доступа и уровню безопасности (например, рядовой пользователь, пользователь-гость, администратор);
- по задачам, которые им приходится решать при выполнении бизнес-операций;
- по используемым функциям;
- по частоте использования продукта;
- по опыту в предметной области и опыту работы с компьютерными системами;
- по используемой платформе (настольные ПК, ноутбуки, планшеты, смартфоны, специализированные устройства);
- по родному языку;
- по виду доступа к системе — прямой или косвенный.

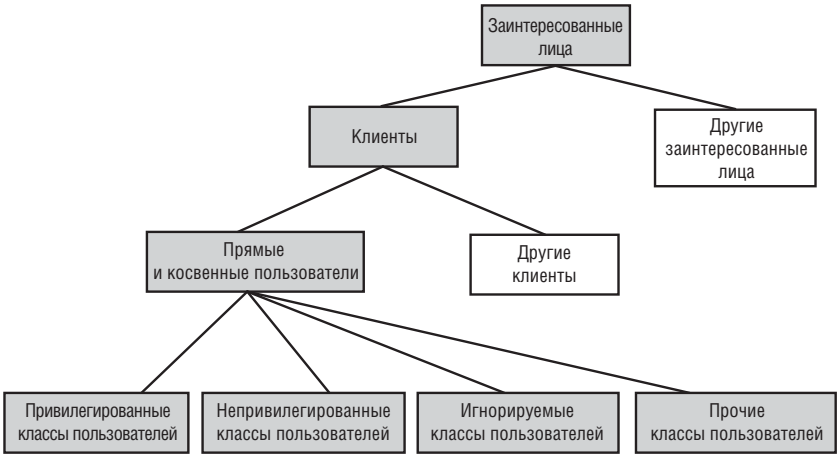


Рис. 6-1. Иерархия клиентов, пользователей и заинтересованных лиц

Очень заманчиво поделить пользователей на классы по их географическому положению или виду бизнеса, которым занимается компания. Например, компания — производитель банковского ПО первоначально предполагала разделить пользователей по типам финансовых учреждений, в которых те работают: крупный коммерческий банк, небольшой коммерческий банк, ссудо-

сберегательное общество или кредитный союз. В действительности же таким образом выделяются потенциальные сегменты рынка, а не различные классы пользователей.

Для определения классов пользователей нужно думать о задачах, которые разные пользователи выполняют в системе. Во всех этих финансовых учреждениях есть операционисты, люди, занимающиеся ссудами, кассиры и т. п. У людей, выполняющих одну работу — потому что так называется их должность или роль — формируются более или менее аналогичные функциональные требования к системе независимо от размера и вида банка, в котором они работают. Операционисты делают одно, ответственные за ссуды другое и т. д. Более логичными названиями для имен классов пользователей в банковской системе будут операционист, специалист по займам и менеджер отделения. Дополнительные классы пользователей можно обнаружить при изучении возможных вариантов использования, пользовательских историй и потоков процессов и тех, кто может их выполнять.

Одни классы пользователей для вас важнее других. Привилегированными называются классы пользователей, удовлетворение которых более всего соответствует достижению бизнес-целей проекта. Когда вы принимаете решения о приоритетах или пытаетесь найти компромисс требований, выдвигаемых различными классами пользователей, мнение привилегированных классов имеет первостепенное значение. Это не означает, что заинтересованных лиц, оплачивающих разработку системы (они, вполне вероятно, вообще не являются ее пользователями), или тех, кто имеет большое политическое влияние, следует обязательно включать в привилегированные классы. Все дело в соответствии бизнес-целям.

Непривилегированные классы составляют те пользователи, которые по юридическим причинам или причинам безопасности, конфиденциальности или правовым причинам не должны работать с продуктом (Gause и Lawrence, 1999). Можно так построить функции, что непривилегированным пользователям будет сложно делать то, что они делать не должны. Примером может служить механизмы безопасности доступа, уровни привилегий пользователей, противовирусные функции (для пользователей не людей) и запись поведения пользователям в журнале. Блокировка учетной записи пользователя после четырех неудачных попыток входа закрывает доступ непривилегированному классу «пользователи, выдающие себя за другого», невзирая на риск создания неудобств забывчивым законным пользователям. Если моему банку не удается распознать компьютер, он отправляет мне сообщение электронной почты с разовым кодом доступа, который я должен ввести перед процедурой входа в систему. Эта функция реализована из-за наличия непривилегированного класса «люди, которые возможно украли мою банковскую информацию».

Остальные классы пользователей можно проигнорировать. Они получат то, что получится, то есть при разработке системы вам не надо учитывать их интересы. Мнение прочих классов пользователей, не относящихся к приви-

легированным или непривилигерованным, при определении требований к продукту имеют примерно одинаковое значение.

У каждого класса пользователей есть свой набор требований, сформировавшийся в ходе выполнения задач. Потребности различных классов пользователей могут перекрываться. Например, возможность проверки баланса счета клиента может требоваться операционистам, менеджерам и специалистам по займам. У разных классов пользователей могут быть разные ожидания качеств, например удобство работы, которые влияют на проектирование пользовательского интерфейса. Новичков волнует, насколько легко научиться работать с продуктом. Такие пользователи предпочитают меню, графические интерфейсы, упорядоченное представление информации на экране, подробные подсказки, мастеры и справочную систему. Тех, кто поопытнее, волнует легкость использования и эффективность работы. Они ценят клавиатурные сокращения, макросы, возможности настройки, панели инструментов, возможности создания сценариев.

> **Внимание!** Не упустите из виду классы косвенных пользователей. Они могут обращаться к вашему приложению не напрямую, а работать с его данными и сервисами через другие приложения или отчеты. Однако даже опосредованный клиент все равно остается вашим клиентом.

Классы пользователей не обязательно состоят из людей. Это могут быть программные агенты, выполняющие обслуживание от имени живого пользователя. Программные агенты могут сканировать сеть в поиске информации о товарах и услугах, получать данные из новостных каналов, обрабатывать входящую электронную почту, наблюдать за физическими системами и сетями на предмет неполадок и вторжений или выполнять интеллектуальный анализ данных (data mining). Интернет-агенты, проверяющие веб-сайтв на предмет уязвимостей или генерирующие спам, являются примером непривилегированного класса пользователей. Обнаружив такие виды непривилегированных классов пользователей, вы можете создать специальные требования не удовлетворять, а предотвращать удовлетворение их потребностей. Например, такие средства веб-сайта, как капча (CAPTCHA), которое призвано отсеять машинных «пользователей», представляющих собой программных агентов, от живых людей за счет предъявления повелителям сайта ввести изображенные на картинке символы, которые легко распознает человек, но практически не в состоянии машина.

Как вы помните, пользователи являются подмножеством клиентов, которые в свою очередь являются подмножеством заинтересованных лиц. Нужно учитывать намного более широкий диапазон возможных источников требований, чем просто классы прямых и косвенных пользователей. Например, хотя члены команды разработчиков не являются конечными пользователями создаваемой вами системы, вам нужна от них информация о внутренних атрибутах качества, таких как эффективность, переносимость и возможность модификации и повторного использования, как описано в главе 14. Одна

компания обнаружила, что каждая установка их продукта представляла собой недешевый кошмар, после чего был создан новый класс пользователей «установщик», который позволил сконцентрироваться на требованиях, таких как разработка архитектуры пользовательской модификации продукта. При обнаружении заинтересованных лиц, от которых может потребоваться информация о требованиях, нужно поискать далеко за рамками набора очевидных конечных пользователей.

Определение классов пользователей

В самом начале работы над проектом необходимо определить и охарактеризовать различные классы пользователей, чтобы можно было выявить у представителей всех важных классов их требования. Один из полезных способов определения классов называется «от большого к малому» («Expand Then Contract») (Gottesdiener, 2002). Начните с опроса куратора на предмет того, кто по его мнению будет использовать систему. Для начала методом мозгового штурма придумайте как можно больше классов пользователей: столько, сколько сможете. Не бойтесь, если их окажется несколько десятков — позже вы объедините их и классифицируете. Важно не пропустить какой-либо класс, иначе это аукнется вам позже. Следующий этап — выявить группы с похожими потребностями: их можно объединить в один класс или рассматривать как несколько подклассов одного крупного класса пользователей. Постарайтесь, чтобы число классов не превышало 15.

Одна компания, разрабатывавшая ПО для примерно 65 корпоративных клиентов, рассматривала каждого из них как отдельного пользователя со своими потребностями. Объединение клиентов в шесть классов значительно упростило работу с требованиями к будущим версиям продукта. Дональд Гаус и Джеральд Уайнберг (Donald Gause и Gerald Weinberg, 1989) предоставляют массу советов по определению возможных пользователей, «обрезке» списка пользователей и поиска конкретных пользователей для участия в проекте.

Различные модели анализа позволяют определить классы пользователей. Внешние сущности, показанные за пределами вашей системы на контекстной диаграмме (см. главу 5), являются кандидатами на классы пользователей candidates for user classes. Структурная схема организации может также помочь обнаружить возможных пользователей и других заинтересованных лиц (Beatty и Chen, 2012). На рис. 6-2 показана часть структурной схемы компании Contoso Pharmaceuticals. Практически все возможные пользователи этой системы скорее всего находятся где-то на этой схеме. В процессе анализа заинтересованных лиц и пользователей изучите структурную схему на предмет:

- отделов, участвующих в бизнес-процессах;
- отделов, на которые оказывают влияние бизнес-процессы;
- отделы и имена ролей, в которых могут находиться прямые или косвенные пользователи;

- классы пользователей, охватывающие несколько отделов;
- отделы, у которых могут быть сношения с внешними заинтересованными лицами за пределами компании.

Схема снижает вероятность пропустить какой-то класс пользователей в организации. Она показывает, где искать возможных представителей конкретных классов пользователей, а также ключевых лиц, ответственных за принятие решений. В одном отделе могут находиться много классов пользователей с различными потребностями. С другой стороны, обнаружение пользователей одного класса во многих отдела может упростить сбор требований. Изучение структурной схемы помогает оценить, со сколькими представителями пользователей вам придется поработать, чтобы быть уверенным, что вы понимаете потребности широкого сообщества пользователей. Также постарайтесь понять, какие типы информации пользователи каждого отдела могут предоставить в зависимости от роли в организации и месте их отдела в проекте.

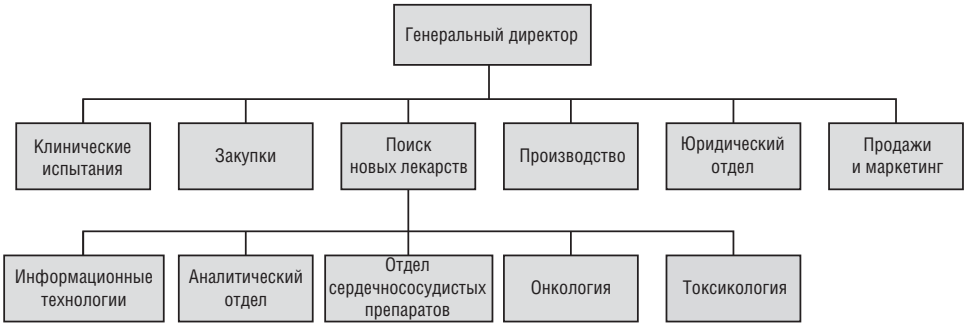


Рис. 6-2. Часть структурной схемы компании Contoso Pharmaceuticals

Задокументируйте классы пользователей и их отличительные черты, обязанности и физическое расположение в спецификации требований к ПО. Сравните эти данные с уже имеющейся информацией профилей заинтересованных лиц в документе концепции и границ во избежание противоречий и дублирования. Включите в спецификацию требований к ПО всю существенную информацию о каждом классе пользователей, например относительный или абсолютный размер и привилегированность класса. Это поможет команде разработчиков определить приоритеты запросов об изменениях и выполнить оценку их влияния. Примерная оценка объема и типа системных транзакций позволит тестировщикам разработать профиль использования системы и в последующем планировать действия по проверке системы. Менеджер проекта и бизнес-аналитик по разработке системы контроля химикатов Chemical Tracking System, о которой шла речь в предыдущих главах, выявил следующие классы пользователей и их характеристики (табл. 6-1).

Табл. 6-1. Классы пользователей в Chemical Tracking System

Имя	Численность пользователей	Описание
Химики (привилегированный)	Примерно тысяча в шести зданиях	Химики посредством системы запрашивают химикаты у поставщиков и со склада. Каждый химик использует систему несколько раз в день, преимущественно для запроса химикатов и контроля за контейнерами, поступающими в лабораторию и отправляемыми из нее. Химикам необходима возможность искать в каталогах поставщиков специальные химические структуры, импортированные из специальных утилит, для создания таких структур
Специалист по закупкам	Пять	Сотрудники отдела закупок обрабатывают запросы химикатов. Они размещают и отслеживают выполнение заказов поставщиками. Они не очень-то разбираются в химии, главное, что им нужно, чтобы поиск в каталогах был максимально простым. Специалистам отдела закупок не нужны возможности отслеживания контейнеров, предоставляемые системой. Каждый из них обращается к системе примерно 25 раз в день
Сотрудники склада химикатов	Шесть техников, один контролер	Сотрудники склада химикатов управляют запасом из более чем 500 000 химических контейнеров. Они обрабатывают запросы от химиков на поставку контейнеров с трех складов, запросы поставщикам на новые химикаты и контролируют поступление контейнеров на склады и их отгрузку. Функция отчета о складских запасах нужна только сотрудникам склада. Из-за большого объема транзакций эта функции, используемые сотрудниками склада должны быть эффективными и автоматизированными
Специалисты отдела охраны труда и техники безопасности (привилегированный класс)	Один менеджер	Специалисты отдела охраны труда и техники безопасности с помощью системы генерируют ежеквартальные отчеты в соответствии с местным и федеральным постановлениям об использовании и утилизации химических веществ. Скорее всего менеджеру этого отдела придется несколько раз в год запрашивать новую форму отчетов, если изменятся требования регулирующих органов. Эти запросы об изменениях имеют высочайший приоритет и должны реализовываться в самые короткие сроки

Возможно, стоит создать каталог классов пользователей, которые повторяются во многих приложениях. Определив такие классы на уровне компании, вы сможете повторно воспользоваться ими в будущих проектах. Вероятно, для следующей системы придется создать несколько новых классов, но скорее всего пригодятся и те, что уже имеются. Если вы решите добавить описания классов пользователей в документ SRS, их можно включить по ссылке, после чего просто добавить все новые группы, характерные для данного проекта.

Архетипы пользователей

Чтобы было легче внедрить идею с классами пользователей в жизнь, стоит описать типичного представителя каждого класса, или *архетип* (persona) пользователя (Cooper, 2004; Leffingwell, 2011). Архетип представляет собой описание гипотетического человека, который является типичным представителем группы пользователей, имеющих похожие характеристики и потребности. Архетипы можно использовать для понимания требований и проектирования поведения интерфейса системы в соответствии с потребностями конкретных сообществ пользователей.

Архетип можно использовать в качестве подстановки, когда у бизнес-аналитика нет под рукой реального представителя пользователя. Чтобы не останавливать работу, бизнес-аналитик может вообразить архетип пользователя, выполняющего определенную задачу, или попытаться определить, каковы предпочтения этого архетипа, — это позволит создать черновой вариант требований, которые можно будет проверить, когда появится реальный пользователь. К параметрам архетипа относятся социальные и демографические характеристики и поведение, предпочтения, раздражающие факторы и тому подобная информация. Надо позаботиться, чтобы архетипы точно представляли соответствующие классы пользователей, основываясь на исследовании рынка и демографических и этнографических исследованиях.

Вот пример архетипа для одного класса пользователя системы Chemical Tracking System:

> *Фред, 41 год, работает химиком в Contoso Pharmaceuticals уже 14 лет, с тех пор как получил степень кандидата наук. Не очень любит компьютеры. Обычно Фред одновременно работает над двумя проектами в смежных областях химии. В его лаборатории хранится около 400 контейнеров и баллонов с химикатами. Ежедневно ему требуется в среднем 4 новых химиката со склада. Два из них — промышленные химикаты из имеющихся на складе, один обычно заказывается и еще один поступает от поставщика специализированных препаратов. Иногда Фреду требуются опасные химикаты, для работы с которыми необходима специальная подготовка. Приобретая какой-либо химикат впервые, Фред хочет, чтобы ему по электронной почте автоматически поступали со-*

ответствующие материалы по технике безопасности. Ежегодно Фред создает около 20 новых химикатов, которые отправляет на склад. Он хочет получать по электронной почте автоматически сгенерированный отчет о заказанных им в прошлом месяце химикатах; это позволит ему контролировать свои контакты с химическими веществами.

Изучая требования химиков, бизнес-аналитик может рассматривать Фреда как архетип данного класса пользователей и задавать себе вопрос: «Что Фреду нужнее всего?» Работа с таким пользователем делает процесс обдумывания требований более осязаемым, чем обдумывание того, что может понадобиться безликой группе людей. Некоторые аналитики даже выбирают случайное человеческое лицо соответствующего пола, чтобы сделать архетип пользователя еще зримее.

Дин Леффингвелл (Dean Leffingwell, 2011) предлагает проектировать систему так, чтобы максимально облегчить человеку, описанному в архетипе, использовать приложение. То есть нужно сосредоточиться на удовлетворении потребностей этого (воображаемого) человека. Если вы создали архетип, точно воспроизводящий класс, это может сильно помочь в выполнении задачи удовлетворения потребностей и ожиданий целого класса. Как сказал один из коллег: «В проекте по обслуживанию торговых автоматов я выдумал мастера по обслуживанию Долли и заведующего складом Ральфа. Мы написали сценарии для них и они стали частью проектной команды, конечно же виртуально».

Представители пользователей

При разработке каждого типа проекта, включая корпоративные информационные системы, коммерческие приложения, встроенные системы, встроенные системы, веб-сайты и заказное ПО, необходимо привлечь к работе представителей пользователей, выражающих мнение большей части клиентов. Они должны участвовать во всех этапах, а не только в одном из начальных — формулировании требований. Ну и, конечно, должны быть представлены все классы пользователей.

Проще всего наладить контакт с пользователями, когда вы создаете внутрикорпоративные приложения. Если вы разрабатываете коммерческое ПО, можно расспросить тех, кто занимается бета-тестированием или «обкаткой» продукта, чтобы выяснить у них требования на начальных этапах работы (подробнее — в разделе «Сторонники продукта, приглашенные со стороны» далее в этой главе). Иногда полезно создать фокус-группы из пользователей для анализа ваших продуктов или ПО конкурирующих компаний. Вместо того чтобы делать предположения, что хотят пользователи, лучше спросить кого-нибудь из них.

Одна компания предложила участникам фокус-группы протестировать различные цифровые фотоаппараты и компьютеры. Как показали результаты, ПО фотоаппаратов этой компании слишком долго выполняло самую

обычную операцию из-за того, что в архитектуру были заложены наименее вероятные варианты использования. После внесения изменений в следующую версию фотоаппарата количество жалоб клиентов на низкую скорость работы уменьшилось.

Убедитесь, что в фокус-группе представлены все классы пользователей, потребности которых должны определять возможности продукта, а также люди с различным опытом: и эксперты, и новички. Если в фокус-группу входят только самые ранние пользователи продукта и любители пофантазировать, вас захлестнет вал сложных и технически трудных требований, которые могут оказаться неинтересными большей части вашей целевой аудитории.

На рис. 6-3 показаны некоторые типичные пути обмена информацией пользователя и разработчика. Как отмечалось в одном исследовании, в наиболее успешных проектах всегда больше каналов общения вообще и тех, что напрямую связывают разработчика и клиента, чем в менее успешных проектах (Keil и Carmel, 1995). Лучше всего, когда разработчики имеют возможность сами пообщаться с нужными пользователями; то есть выполняют роль бизнес-аналитиков. Это может работать в очень маленьких проектах, при условии, что разработчик обладает соответствующими навыками бизнес-анализа, но не масштабируется до крупных проектов с тысячами возможных пользователей и десятками разработчиков.

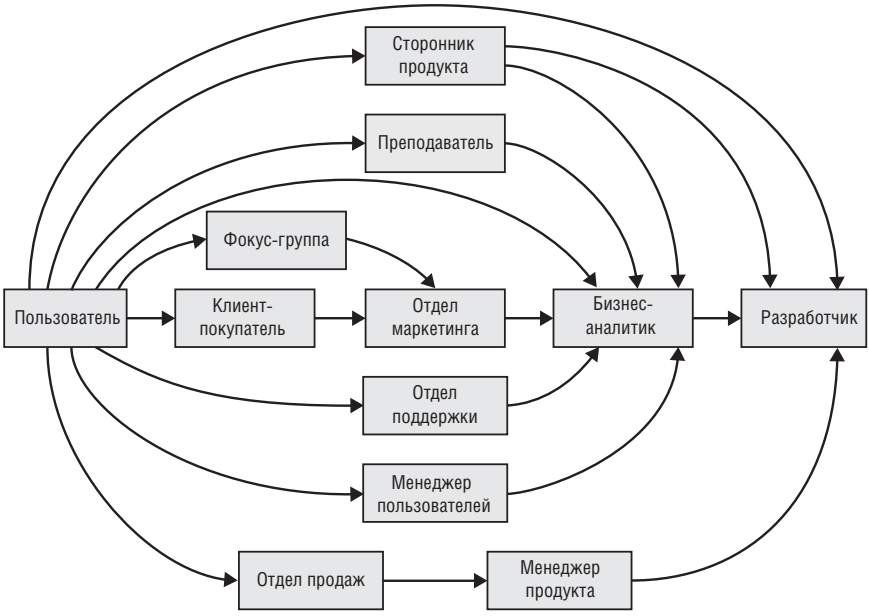


Рис. 6-3. Некоторые возможные каналы обмена информацией между пользователями и разработчиками

Как и в детской игре «Испорченный телефон», дополнительные промежуточные звенья увеличивают вероятность непонимания. Но некоторые из этих промежуточных звеньев могут приносить пользу, например опытный бизнес-

аналитик или другой участник, работающий с пользователями над сбором, оценкой, уточнением и организацией полученной от них информации. Нужно понимать риск, на который вы идете, используя в качестве представителей реального мнения клиента персонал отдела маркетинга, менеджеров продукта, профильных специалистов или каких-либо других лиц. Несмотря на препятствия и затраты, связанные с поиском и выбором представителей пользователей, ваш продукт и клиенты пострадают, если вы пренебрежете общением с людьми, способными предоставить наиболее качественную информацию.

Сторонник продукта

Много лет назад я работал в небольшой группе разработки программных продуктов, которая обеспечивала поддержку научных исследований в крупной корпорации. Во всех наших проектах принимали участие несколько наиболее активных представителей пользователей, которые помогали формулировать требования. Мы называли их *сторонниками продукта* (product champion), или приверженцами проекта (Wiegers, 1996). Привлечение искренне заинтересованных в продукте пользователей — эффективный способ структурировать очень важное партнерство клиентов и разработчиков, о котором идет речь в главе 2.

Каждый сторонник продукта — это основной посредник между членами определенного класса пользователей и бизнес-аналитиком. В идеале сторонники должны быть реальными пользователями, а не суррогатными, как, например кураторы, специалисты по маркетингу, менеджеры или разработчики ПО, которые представляют себя на месте пользователя. Их задача — выяснить потребности остальных членов класса пользователей, который они представляют, и согласовать их. Таким образом, разработка требований — это общее дело аналитика и выбранных клиентов; хотя документацию составляет все-таки аналитик. Сложно написать хорошие требования, если это не ваша основная профессия; нереалистично ожидать, что пользователи, ранее никогда не писавшие требования, хорошо справятся с этой задачей.

Лучшие из сторонников продукта имеют четкое представление о новой системе и в полной мере увлечены ею, так как понимают, какие преимущества она даст им и их коллегам. Сторонник продукта должен уметь отлично общаться и пользоваться авторитетом в компании. Кроме того, необходимо, чтобы он разбирался в предметной области и рабочей среде приложения. Такие сотрудники весьма заняты своей основной работой, поэтому вам необходимо продумать очень убедительные доводы, дабы обосновать крайнюю важность именно этих людей для успеха проекта. Например, сторонники продукта могут возглавить принятие приложения сообществом пользователей, что может оказаться критерием успеха, который так ценят менеджеры. Мы убедились, что хорошие сторонники делают очень много для успеха проектов, и мы всячески поощряем их участие, выражая им нашу признательность и поддержку.

Мы рады отметить и еще одно преимущество от сотрудничества со сторонниками продукта. В нескольких наших проектах участвовали отличные ребята, которые от нашего лица общались с коллегами и клиентами, когда те возмущались тем, что проект до сих пор не завершен. «Не волнуйтесь, — говорили они коллегам и начальству. — Я понимаю и согласен с тем, как команда разработчиков подходит к делу. Время, которое мы потратим сейчас на работу над требованиями, поможет нам получить именно ту систему, которую мы хотим, и сэкономит время в дальнейшем». Такое сотрудничество помогает устранить напряжение, зачастую возникающее между клиентами и командой разработчиков.

Наилучшие результаты отмечены, если каждый сторонник продукта обладает всеми полномочиями для принятия необходимых решений от имени класса пользователей, который он представляет. Если решения сторонника продукта регулярно отвергаются менеджерами или группой разработчиков, он попусту тратит свое время и добрую волю. Тем не менее, горячим сторонникам продукта следует помнить, что они — не единственные клиенты. Проблемы возникают, если тот, кто играет такую важную роль посредника, общается с коллегами не должным образом и выражает только собственные пожелания и идеи.

Сторонники продукта, приглашенные со стороны

При разработке коммерческого ПО иногда трудно найти сторонников продукта вне компании. В некоторых случаях компании-разработчики коммерческого ПО полагаются на собственных профильных специалистов или внешних консультантов, подменяющих реальных пользователей, о которых может быть ничего не известно или которых трудно привлечь к работе над проектом. Если у вас налажены тесные деловые отношения с какими-либо крупными корпоративными клиентами, они могут благосклонно воспользоваться возможностью поучаствовать в формировании требований. Сторонников продукта, приглашенных со стороны, можно заинтересовать экономически, например предложить им скидку на продукт или оплатить их работу над требованиями. Тем не менее, здесь существует некоторая опасность: вам придется научиться слушать голоса не только ярых приверженцев, чтобы не упустить мнения других клиентов. При наличии разнородной клиентской базы сначала нужно определить основные требования, общие для всех клиентов, затем дополнительные требования, важные для отдельных клиентов, сегментов рынка или классов пользователей.

Другой вариант — нанять подходящего сторонника продукта с соответствующим опытом. Одна компания, создававшая систему с центральным сервером и клиентскими приложениями для точек розничной торговли, пригласила в качестве сторонников продукта на полный рабочий день трех менеджеров магазина. Вот еще один пример: Арт, мой давний семейный доктор, оставил врачебную практику и устроился в компанию, разрабатывающую ме-

дицинское ПО, где представлял мнение и интересы врачей. Работодатель Арта был уверен: штатный сотрудник, имеющий врачебный опыт, поможет компании разрабатывать ПО, которое придется по душе другим докторам. Третья компания пригласила на работу нескольких бывших сотрудников одного из своих основных клиентов. Эти люди поделились ценным опытом в предметной области и описали политику организации-клиента. Проиллюстрирую другую модель участия: у одной компании было несколько корпоративных клиентов, активно использующих свои системы выписки счетов. Компания не стала привлекать сторонников со стороны клиентов, а отправила бизнес-аналитиков в подразделения клиента. Клиенты с удовольствие посвятили часть времени своих сотрудников на то, чтобы помочь бизнес-аналитикам собрать корректные требования к новой системе выписки счетов.

Если сторонник продукта — бывший пользователь или увлеченно играет роль пользователя, таковым не являясь, опасайтесь, что его восприятие проблем и потребности реальных пользователей могут различаться. Некоторые предметные области меняются очень быстро, другие относительно стабильны. В любом случае, люди, которые давно в реальности вы выполняли определенную роль, просто забывают детали повседневной работы. Основной вопрос в том, сможет ли сторонник продукта, независимо от его опыта и настоящей должности, точно отражать потребности реальных пользователей.

Чего следует ожидать от сторонника продукта

Чтобы сотрудничество со сторонниками продукта оказалось успешным, задокументируйте, что именно, по-вашему, они должны делать. Эти письменные свидетельства помогут вам полнее выразить ожидания конкретных людей. В табл. 6-2 перечислены некоторые обязанности, которые может выполнять сторонник продукта (Wiegers, 1996). Не каждый станет выполнять их все; используйте эту таблицу как отправную точку для обсуждения с каждым сторонником продукта его обязанностей.

Табл. 6-2. Возможные обязанности сторонника продукта

Категория	Действия
Планирование	Уточнение границ и ограничений продукта
	Определение интерфейсов для работы с другими системами
	Оценка влияния новой системы на бизнес-операции
	Разработка путей перехода со старых приложений или от ручных операций
	Определение применимых стандартов и сертификационных требований

Табл. 6-2. (*окончание*)

Категория	Действия
Требования	Сбор требований от других пользователей
	Разработка сценариев и вариантов использования продукта, а также пользовательских историй
	Разрешение конфликтов между предложенными требованиями в рамках классов пользователей
	Определение приоритетов реализации
	Определение атрибутов качества и производительности
	Оценка прототипов
	Сотрудничество с ответственными за принятие решений над разрешением противоречий требований, поступивших от разных заинтересованных лиц
	Предоставление специализированных алгоритмов
Проверка и утверждение	Рецензирование документов с требованиями
	Определите критериев приемлемости
	Разработка приемочных тестов на основе сценариев использования
	Создание наборов тестовых данных на основе бизнес-данных
	Выполнение бета-тестирования и приемочных испытаний пользователями
Помощь пользователям	Написание фрагментов руководств пользователя и справочных систем
	Подготовка части материалов для учебных пособий
	Демонстрация системы коллегам
Управление изменениями	Оценка исправлений дефектов и предложений об усовершенствовании системы и определение порядка их реализации
	Динамическая корректировка границ будущих выпусков и итераций
	Оценка того, как предполагаемые изменения требований повлияют на пользователей и на бизнес-процессы
	Участие в принятии решений о внесении изменений

На что способны несколько сторонников продукта

Один человек вряд ли сможет описать потребности всех пользователей приложения. Для системы контроля химикатов Chemical Tracking System определено четыре основных класса пользователей, и поэтому для нее следует выбрать четырех сторонников продукта из числа сотрудников Contoso Pharmaceuticals. На рис. 6-4 показана структура команды, которую менеджер проекта организовал из аналитиков и сторонников продукта для сбора кор-

ректных требований из корректных источников. Сторонники не привлекались на полный рабочий день, каждый из них уделял проекту несколько часов в неделю. Три аналитика совместно с четырьмя сторонниками выявляли, анализировали и документировали их требования (один из аналитиков работал с двумя сторонниками, поскольку классы пользователей «Специалист по закупкам» и «Сотрудник отдела охраны труда и техники безопасности» невелики и выдвинули всего несколько требований). Один из аналитиков сводил все требования в единую спецификацию требований к ПО.

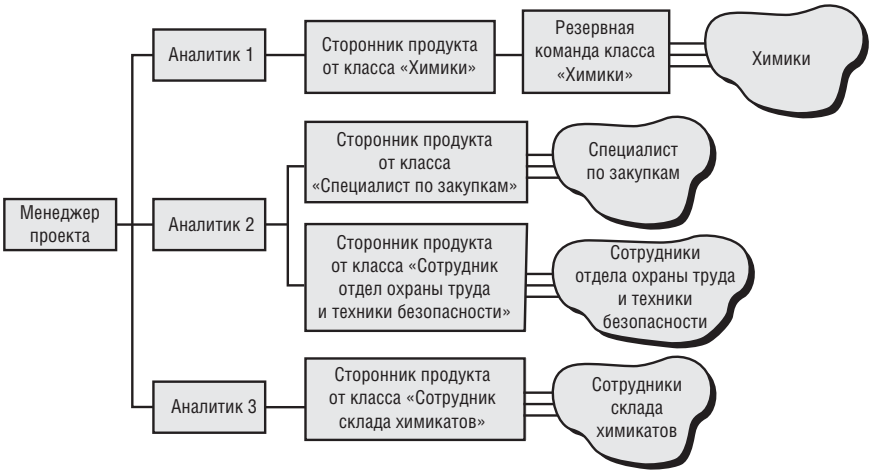


Рис. 6-4. Структура команды из аналитиков и сторонников продукта для системы Chemical Tracking System

Один человек не может в полной мере представить все разнообразные требования крупного класса пользователей, например нескольких сотен химиков Contoso. Чтобы помочь ему, Дон, сторонник продукта от класса «Химики», собрал резервную команду из пяти химиков, работающих в других подразделениях компании. Эти сотрудники представляют подклассы обширного класса пользователей «Химики». Такой иерархический подход позволил вовлечь в разработку требований дополнительных пользователей и в то же время сэкономить на проведении множества круглых столов и десятков интервью. Дон, всегда стремился к консенсусу, но в тех случаях, когда соглашения достичь не удавалось, он без колебаний принимал необходимые решения, чтобы работа не буксовала. Необходимость в резервной команде отпадает, если класс пользователей невелик или сплочен настолько, что интересы группы может в полной мере представлять один человек.[1]

[1] У этой истории интересное завершение. Годы спустя после того, как я завершил работу в этом проекте, один из моих слушателей сказал, что работал в компании, которую Contoso Pharmaceuticals наняла для разработки системы Chemical Tracking System. Разработчики обнаружили, что спецификация требований, которую мы создали с использованием этой модели работы со сторонниками продукта, стала надежным основанием для разработки. Система была успешно создана и много лет использовалась в Contoso.

Безголосый класс пользователей

Бизнес-аналитик в компании Humongous Insurance был очень доволен, узнав, что пользующийся авторитетом сотрудник, Ребекка, согласилась стать сторонником продукта для новой системы обработки исков. У Ребекки было много идей насчет возможностей и пользовательского интерфейса системы. Вдохновленная возможностью работать под руководством эксперта, команда разработчиков с удовольствием реализовала ее предложения. Однако, выпустив продукт, они были шокированы количеством жалоб на трудности работы с системой.

Ребекка — опытный пользователь. Предложенные ею функции, обеспечивающие удобство работы с системой, хороши для таких же опытных пользователей, как и она сама, однако 90% клиентов, *гораздо менее* опытных, нашли, что система интуитивно непонятна и трудна для освоения. Бизнес-аналитик вовремя не понял, что у системы обработки исков по крайней мере два класса пользователей. Большая группа начинающих пользователей оказалась лишенной права на участие в разработке требований и пользовательского интерфейса, поэтому компании Humongous пришлось потратиться на переделку проекта, а она обошлась недешево. Аналитику следовало привлечь второго сторонника продукта — выразителя интересов большого класса неопытных пользователей.

Как «продать» идею о необходимости привлечения сторонника продукта

Будьте готовы встретить сопротивление, когда вы предложите привлечь к работе над проектом сторонников продукта. «Пользователи слишком заняты». «Менеджеры хотят сами принимать решения». «Они затормозят нашу работу». «Мы не можем себе этого позволить». «Я не знаю, что могу предложить в качестве сторонника продукта». Некоторые пользователи не хотят высказывать свои требования к системе, которая, как они думают, заставит их изменить методы работы или даже создаст угрозу увольнения. Иногда менеджеры неохотно делегируют обычным пользователям полномочия, касающиеся требований.

Отделив бизнес-требования от требований пользователей, вы смягчите эти препятствия. Будучи реальным пользователем, сторонник продукта принимает решения на уровне интересов пользователей в рамках, налагаемых бизнес-требованиями проекта. Куратор сохраняет всю полноту власти, чтобы принимать решения, касающиеся вида продукта, границ проекта, графика работы и затрат. Документируя и обсуждая роль и обязанности сторонника продукта, вы даете предполагаемым кандидатам на эту роль возможность уверенно отвечать на вопрос, чем же они занимаются. Напомните руководству, что сторонник продукта является ключевым источником информации, который поможет достичь в проекте поставленных бизнес-целей.

Если же вы столкнетесь с сопротивлением, укажите, что недостаточное участие пользователей — общепризнанная основная причина провала проектов по разработке ПО. Напомните оппонентам о проблемах предыдущих проектов, вызванных недостаточным вовлечением пользователей в работу. Каждая компания хранит свои страшилки о новых системах, которые не удовлетворили потребности пользователей, или оказались не столь полезны, или не оправдали ожидания. Вы не можете позволить себе переделывать системы или отказываться от них, если они не соответствуют необходимым требованиям только потому, что никто не разобрался в требованиях. Привлечение сторонников — один из способов получить эту важную информацию от пользователей вовремя, а не в конце проекта, когда клиенты разочарованы и разработчики истощены.

В какие ловушки можно угодить, привлекая сторонников продукта

Модель привлечения сторонников продукта успешно показала себя во многих рабочих средах. Она имеет смысл, только если сторонник понимает и принимает свои обязанности, обладает полномочиями для принятия решений на уровне требований пользователей и имеет время для выполнения данной работы. Остерегайтесь следующих возможных проблем.

- Некоторые менеджеры изменяют решения, принятые квалифицированным и официально назначенным сторонником продукта. Возможно, у менеджера в последнюю минуту появилась новая безумная идея, он из прихоти захотел изменить направление работы, или думает, что знает все потребности пользователей. Зачастую результат такого поведения — недовольные пользователи и расстроенные сторонники продукта, чувствующие, что руководство им не доверяет.
- Сторонник продукта, забыв, что он представляет других клиентов, озвучивает только собственные требования: конечно же, ему не удастся хорошо выполнять свои обязанности. Возможно, лично ему результат понравится, а вот остальным — вряд ли.
- Сторонник продукта, не имеющий четкого представления о новой системе, может уступить решение важных вопросов аналитику. Если все идеи аналитика вызывают одобрение сторонника, его трудно назвать полноценным помощником.
- Из-за нехватки времени опытный пользователь назначает сторонником продукта менее квалифицированного коллегу. Это может привести к давлению со стороны опытного пользователя, который по-прежнему желает направлять ход работы над проектом.

Остерегайтесь пользователей, пытающихся выступить от имени класса, к которому не относятся. Иногда один из участников может активно стараться не позволить бизнес-аналитику работать с идеально подходящим пользователем. В случае с системой контроля химикатов компании Contoso сторонник

продукта от сотрудников склада химикатов настаивала, что может выразить требования класса химиков, но, к сожалению, предоставленная ей информация была неточной. Ее было весьма трудно убедить, что это не ее работа, однако аналитик не позволил ей оказывать давление. Менеджер проекта пригласил сторонника от класса химиков, и тот проделал блестящую работу по сбору, оценке и передаче требований этих специалистов.

Представительство пользователей в проектах гибкой разработки

Эффективное общение между членами проектной команды и соответствующими клиентами — наиболее эффективный способ разрешения многих проблем с требованиями и дополнения специфическими деталями требований при необходимости. Сколь детальной ни была документация на бумаге, она все равно не заменяет общения в процесс проекта. Базовая основа экстремального программирования, одного из первых методик гибкой разработки (agile), — постоянное наличие клиента на месте для подобного общения (Jeffries, Anderson и Hendrickson, 2001).

Некоторые методы гибкой разработки включают наличие в команде одного представителя заинтересованных лиц, который называется *владельцем продукта* (product owner) и выступает от лица клиента (Schwaber, 2004; Cohn, 2010; Leffingwell, 2011). Владелец продукта определяет концепцию продукта и отвечает за разработку и приоритеты содержимого резерва продукта. (*Резерв* (backlog) представляет собой упорядоченный по приоритетам список пользовательских историй, то есть требований, для продукта и их распределение по будущим итерациям, которые в методологии гибкой разработки Scrum называются спринтами.) Таким образом, владелец продукта отвечает за все три уровня требований: пользовательские, функциональные и бизнес-требования. По сути он объединяет функции сторонника продукта и бизнес-аналитика, представляя клиента, определяя функции продукта и их приоритеты и т. п. В конце концов кто-то должен принимать решения, какую функциональность реализовывать в продукте и когда. В Scrum за это отвечает владелец продукта.

Идеальный вариант наличия одного владельца продукта не всегда практичен. Я знал одну компанию, в которой реализовывали пакетное решение для обслуживания страховых операций. Организация была слишком большой и сложной, чтобы нашелся человек, который понимал все достаточно глубоко и подробно, чтобы принимать все решения в процессе реализации. Клиенты решили выбрать владельцев продукта из каждого отдела, которые отвечали за приоритеты функциональности в своем отделе. Директор по ИТ служил ведущим среди владельцев продукта. Он владел концепцией всего продукта и мог обеспечить, чтобы отделы не отступали от нее. Он нес ответственность за принятие решений в случае возникновения противоречий между владельцами продукта на уровне отделов.

Присутствующее в гибкой методике положение о постоянном наличии клиента на месте и тесном сотрудничестве разработчиков с клиентом является очень разумным. На самом деле нам бы очень хотелось, чтобы во *всех* проектах по разработке делалось такое ударение на участии пользователей. Но как вы видели, во всех отличных от очень маленьких проектов есть не один класс пользователей, а также много других заинтересованных лиц, интересы которых должны быть представлены в продукте. Во многих случаях нереалистично ожидать, что один человек сможет адекватно понять и описать потребности всех нужных классов пользователей, не говоря уже о принятии всех решений, связанных с определением продукта. Особенно во внутрикорпоративных проектах обычно лучше использовать структуры представительства, такие как модель сторонников продукта, чтобы обеспечить адекватное участие пользователей.

Схемы владельца продукта и сторонника продукта не являются взаимоисключающими. Если владелец продукта выступает в роли бизнес-аналитика, а не представителя заинтересованного лица, он может создать структуру из одного или большего числа сторонников продукта, предоставляющих самую корректную информацию. Другой подход заключается в сотрудничестве владельца продукта с одним или несколькими бизнес-аналитиками, которые в свою очередь работают с заинтересованными лицами над требованиями. В такой ситуации владелец продукта выступает главным ответственным за принятие решений.

Важность наличия клиента на месте

Однажды мне пришлось писать программы для научного сотрудника, место которого находилось на расстоянии трех метров от моего. Джон мог предоставлять моментальные ответы на мои вопросы, выражать свое мнение о дизайне пользовательского интерфейса и уточнять наши зафиксированные в письменном виде требования. В одни прекрасный деть Джон переехал в новое помещение на том же этаже за углом на расстоянии примерно 30 метров. Я сразу же ощутил снижение своей продуктивности как программиста, потому что возросли задержки в цикле общения с Джоном. Я тратил больше времени на устранение недостатков, потому что часто долго двигался в неправильном направлении, не имея возможности вовремя скорректировать курс. Ничем нельзя заменить постоянное присутствие клиентов не просто на месте, но и в непосредственной близости. Но при этом надо остерегаться слишком частых прерываний, которые отвлекают людей, не позволяя им полностью сосредоточиться на своей работе. Иногда требуется четверть часа, чтобы обратно погрузиться в продуктивное состояние ума (DeMarco и Lister, 1999).

Наличие клиента на месте не гарантирует получение требуемого результата. Мой коллега менеджер проекта Крис создал для команды среду разработки с минимальными физическими барьерам и привлек двух сторонников

продукта. Крисс рассказал о результатах так: «Похоже, что близкое расположение хорошо работает в команде разработчиков, а в среде сторонников продукта результаты неоднозначные. Один из них сидел в самой середине и все равно умудрялся избегать нас всех. Новый сторонник отлично ладит с разработчиками и оказал реальную помощь в ускорении разработки ПО». Ничто не заменит наличие правильных людей, в правильной роли, в правильном месте и с правильным отношением.

Разрешение противоречивых требований

Кто-то должен устранять конфликты между требованиями разных классов пользователей, сглаживать противоречия и выступать арбитром при решении вопросов, возникающих относительно границ проекта. Во многих, но не во всех случаях это могут делать сторонники или владелец продукта. На ранних стадиях проекта необходимо определить лиц, которые будут принимать решения, касающиеся требований, как об этом говорилось в главе 2. Если не ясно, кто за это отвечает, или уполномоченные лица отказываются брать на себя такую ответственность, обязанность принимать решения по умолчанию возлагается на разработчиков или аналитика. Но у большинства из них нет нужных знаний или кругозора для принятия оптимальных бизнес-решений. Аналитики иногда прислушиваются к наиболее громогласному сотруднику или мнению начальства. Это понятный, но не лучший выбор. Решения должны принимать сотрудники на низшем уровне организационной иерархии, которые непосредственно решают задачи.

В таблице 6-3 перечислены некоторые конфликты, которые могут возникнуть в ходе работы над проектами, а также способы их устранения. Ведущим проект сотрудникам необходимо определить, кто будет принимать решения при возникновении подобных ситуаций, кто обладает правом голоса, если согласие не достигнуто, и к кому нужно апеллировать при необходимости.

Табл. 6-3. Предложения по разрешению конфликтов при определении требований

Разногласия между…	Как разрешать
...отдельными пользователями	Решение принимает сторонник или владелец продукта
...классами пользователей	Предпочтение отдается наиболее важному классу пользователей
...сегментами рынка	Предпочтение отдается сегменту, оказывающему наибольшее влияние на успех бизнеса
...корпоративными клиентами	Решение определяется на основе бизнес-целей
...пользователями и менеджерами пользователей	Решение принимает сторонник или владелец продукта класса пользователей
...разработчиками и клиентами	Предпочтение отдается клиентам, но с учетом бизнес-целей
...разработчиками и специалистами по маркетингу	Предпочтение отдается специалистам по маркетингу

Внимание! Не оправдывайте любые запросы клиента только потому, что «клиент всегда прав». Мы все знаем, что клиент прав *не всегда* (Wiegers, 2011). Клиент бывает нерациональным, плохо информированным или в дурном настроении. Однако у него есть своя точка зрения, и команда разработчиков должна понимать и уважать ее.

Переговоры о требованиях не всегда проходят так, как они должны проходить по мнению аналитиков. Нам приходилось наблюдать случаи, когда специалисты по маркетингу никогда не отказывали клиентам в их запросах, несмотря на их нереалистичность и дороговизну. Команде следует определить, кто будет принимать решения по требованиям к проекту, еще до начала подобных столкновений. В противном случае нерешительность и пересмотр уже принятых решений приведут к тому, что работа над проектом заглохнет в бесконечных спорах. Если вы бизнес-аналитик, попавший в подобные жернова, используйте структуру и процессы организации для разрешения таких разногласий. Но, как мы уже говорили ранее, нет простых решений, если приходится иметь дело с неблагоразумными людьми.

Что дальше?

- Сравните схему на рис. 6-1 с тем способом, посредством которого вы узнаете мнение клиентов. Возникают ли у вас проблемы с текущими информационными каналами? Определите кратчайшие и самые эффективные пути коммуникации, которые можно использовать для сбора требований в будущем.
- Определите для своего проекта различные классы пользователей. Какие них привилегированные, а какие (если таковые имеются) — нет? Кто бы мог стать хорошим сторонником продукта для каждого из важных классов пользователей? Даже если проект уже выполняется, команда скорее всего выиграет от наличия сторонников продукта.
- Используя табл. 6-2 в качестве отправной точки, определите обязанности сторонников продукта. Обсудите конкретные пункты с каждым предполагаемым сторонником продукта и его менеджером.
- Определите, кто должен принимать решения по вопросам, связанным с требованиями к проекту. Насколько эффективен процесс принятия решений, используемый в настоящее время? Где он дает сбой? Правильно ли выбраны лица, принимающие решения? Если нет, кто должен принимать решения? Предложите лицам, принимающим решения, последовательность действий для достижения согласия по вопросам требований.

Глава 7

Выявление требований

«Доброе утро, Мария. Я — Фил, аналитик требований к новой информационной системе для сотрудников, которую мы собираемся разрабатывать для вас. Спасибо, что согласились стать сторонником продукта этого проекта. Ваша информация нам очень пригодится. А теперь расскажите мне, что же вам все-таки нужно?»

«Гм, что же мне нужно, — размышляет Мария. — Не представляю, с чего и начать. Ну, новая система должна работать намного быстрее предыдущей. Потом, вы же знаете, как старая система зависает, если имя какого-либо сотрудника оказывается слишком длинным — нам приходится обращаться в службу поддержки, чтобы они ввели имя. В новой системе длинные имена должны обрабатываться без сбоев. Да, согласно новому закону, нам нельзя использовать номера социального страхования в качестве идентификаторов сотрудников, и с вводом новой системы в эксплуатацию нам придется заменить все эти данные. Также пригодилась бы функция отчета о времени, затраченном каждым сотрудников в текущем году на обучение.

Фил добросовестно записал все пожелания Мария, но его голова пошла кругом. Он понятия не имел, насколько потребности Марии соответствуют бизнес-целям проекта. Он не знал, что делать с этой информацией и какие вопросы задавать.

Сердце процесса формулирования требований — их *выявление* (elicitation), когда определяются потребности и ограничения лиц, заинтересованных в программной системе. Выявление не то же, что «сбор требований». И это не точная запись того, что сказали пользователи. Выявление требований — совместный и аналитический процесс, включающий такие действия, как сбор, обнаружение, извлечение и определение требований. Выявление применяется для обнаружения пользовательских, функциональных, нефункциональных и бизнес-требований, а также других видов информации. Выявление требований, наверное, самая сложная, критически важная, чреватая ошибками и требующая много общения сторона разработки ПО.

Вовлечение пользователей в процесс сбора информации — это способ получить поддержку проекта. Если вы бизнес-аналитик, попытайтесь понять, из чего проистекают конкретные требования пользователей. Последовательно

изучите работу пользователей, чтобы принять решения об этой работе, и постарайтесь понять ее логику. Убедитесь, что все понимают, почему система должна выполнять те или иные функции. Иногда предложенные требования отражают устаревшие или неэффективные бизнес-процессы, которые *не следует* встраивать в новую систему.

Аналитик должен создать атмосферу, благоприятную для тщательного исследования обусловленного продукта. Чтобы облегчить общение, используйте лексику предметной области, а не прессингуйте клиентов компьютерным жаргоном. Не полагайтесь на то, что все участники одинаково понимают важные термины предметной области, — создайте словарь. Однако объясните клиентам, что обсуждение возможной функциональности — это еще не обязательство включить ее в продукт. Этап мозгового штурма и анализ возможностей обособлен от анализа приоритетов, осуществимости и ограничений. Заинтересованным в проекте лицам необходимо расставить приоритеты в списке «голубых грез», чтобы проект не стал безразмерным и бесполезным.

Результат этапа формулирования требований — согласованное представление о потребностях всех заинтересованных в проекте лиц. Когда разработчики понимают эти потребности, им гораздо легче предлагать альтернативные способы их удовлетворения. Те, кто занимается выявлением требований, не должны поддаваться искушению немедленно приступить к проектированию системы — до того момента, как проблема станет абсолютно ясной. В противном случае готовьтесь к значительной переработке проекта по мере детализации требований. Концентрация на задачах пользователей, а не на интерфейсе, внимание к ключевым потребностям, а не к пожеланиям позволяет команде преждевременно не отвлекаться на детали архитектуры.

Как показано на рис. 7-1, процесс разработки цикличен по своей природе. Выявляется часть требований, после чего анализируется обнаруженная информация, формулируется часть требований, обнаруживается, что какой-то информации не хватает, выполняется дополнительное выявление требований и т. д. Не стоит ожидать, что вы проведете ряд семинаров по выявлению требований, объявите о победе и все — можно заниматься другим проектом.

Рис. 7-1. Цикл выявления, анализа и спецификации требований

В этой главе рассказывается о разных эффективных методах выявления требований, а также о сложностях и рекомендациях, когда применять тот или иной метод. В остальной части главы описывается процесс выявления требований в целом — от планирования до организации получения результатов

встречи. Далее в главе мы предупреждаем о некоторых подводных камнях процесса выявления требований, а также предоставляем советы, как выявлять отсутствующие требования. На рис. 7-2 показаны действия одной отдельно взятой встречи по выявлению требований. Прежде чем мы начнем изучать этот процесс, познакомимся с некоторыми методам выявления требований, которые могут оказаться полезными.

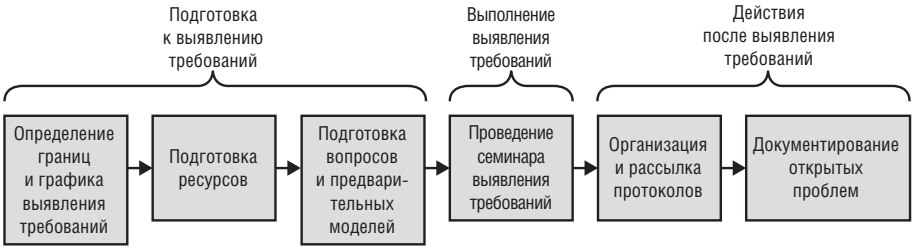


Рис. 7-2. Последовательность действий во встрече по выявлению требований

Методы выявления требований

В проектах по разработке ПО могут применяться разные методы выявления требований. На самом деле, вряд ли найдется проектная команда, в которой используется только один метод. Всегда есть несколько типов информации, которую надо выявлять, и разные заинтересованные лица предпочитают разные подходы. Один пользователь может четко сформулировать, как он использует систему, а за другим придется понаблюдать в работе, чтобы получить такое понимание сценария использования.

Методы выявления требований делятся на коллективные, в которых участвуют заинтересованные лица, и независимые, когда вы работаете сами над выявлением информации. Коллективные методы ориентируются на выявление пользовательских и бизнес-требований. Непосредственная работа с пользователями необходима, потому что пользовательские требования связаны с задачами, которые пользователи должны выполнять в системе. Для выявления бизнес-требований нужно работать с людьми, такими как куратор проекта. Независимые методы дополняют требования, предоставляемые пользователями, и позволяют выявить функциональность, о которой конечные пользователи могут не знать. В большинстве проектов используется сочетание коллективных и независимых методов выявления требований. Разные методы предоставляют возможность по-разному исследовать требования и даже могут выявлять разные требования. В следующих разделах описывается несколько методов, обычно применяемых для выявления требований.

Интервью

Самый очевидный способ узнать, что нужно пользователям системы, — просто спросить у них. Интервью — традиционный источник требований как для

серийных продуктов, так и информационных систем в любых методиках разработки ПО. Большинство бизнес-аналитиков организуют в какой-то форме индивидуальные интервью или интервью в небольших группах для выявления требований в своих проектах. В проектах гибкой разработки (agile) интервью активно используются в качестве механизма непосредственного привлечения пользователей. Интервью проще запланировать и провести, чем крупные групповые мероприятия, такие как семинары по выявлению требований.

Если вы новичок в предметной области, интервью с экспертами могут помочь быстро войти в курс дела. Это может позволить подготовить предварительные требования и модели для использования в других интервью и семинарах. Если наладить доверительные отношения с интервьюируемыми, они легче делятся своими мыслями в формате «один на один» или в небольшой группе, чем на крупном семинаре, особенно если речь идет о деликатных вещах. В таком «камерном» формате также легче добиться от пользователя поддержки проекта или проверки требований. Интервью также удобны для сбора бизнес-требований у топ-менеджеров, у которых обычно мало времени на встречи.

Есть другие источники, более подробно описывающие, как интервьюировать пользователей (Ian Alexander и Ljerka Beus-Dukic, 2009; Howard Podeswa, 2009). Далее приводятся несколько рекомендаций по проведению интервью. Есть также полезные советы по проведению семинаров по выявлению требований.

Установите контакт Вначале интервью представьтесь, если участники вас еще не знают, изложите план интервью, напомните участникам о целях данного мероприятия и ответьте на все предварительные вопросы и развейте беспокойство, которое может быть у участников.

Придерживайтесь границ проекта Как и на любом другом семинаре по выявлению требований следите, чтобы дискуссия не уходила от основной цели. Даже если вы разговариваете лишь с одним человеком или маленькой группой, есть шанс, что обсуждение уйдет в сторону.

Заранее подготовьте вопросы и предварительные модели Подготовьтесь к интервью, написав все предварительные материалы, которые можно написать заранее, например сформулируйте список вопросов, которые будете задавать в процессе интервью. Предварительные материалы дадут пользователям точку отсчета, с которой можно начать разговор. Людям намного проще давать замечания и критиковать материал, чем создавать новый. Подготовка вопросов и предварительных моделей описаны далее в этой главе в разделе «Подготовка к выявлению требований».

Предлагайте идеи В процессе сбора информации работающий творчески аналитик не только фиксирует слова клиента, но и подкидывает пользователям новые идеи и предлагает альтернативы. Иногда пользователи не представляют, какие возможности готовы предоставить разработчики; они страшно обрадуются, если вы предложите функциональность, которая сделает систему особенно полезной. В тех случаях, когда пользователи не спо-

собны ясно выразить свои потребности, аналитику стоит понаблюдать за их работой, чтобы самому разобраться, какие операции следует автоматизировать (см. раздел «Наблюдение» далее в этой главе). Отлично, когда аналитикам удается выйти за рамки, ограничивающие мышление людей слишком тесно связанных с конкретной предметной областью.

Слушайте активно Применяйте приемы, демонстрирующие активное слушание (наклоняйтесь вперед, проявляйте терпение, предоставляйте вербальную обратную связь и спрашивайте, если что-то неясно), и перефразируйте рассказчика (повторно излагайте основную идею излагаемого рассказчиком, чтобы показать свое понимание).

Семинары

Семинары способствуют совместной работе заинтересованных лиц над определением требований. Эллен Готтендинер (Ellen Gottesdiener, 2002) определяет семинар по выявлению требований как «структурированное совещание, на котором специально отобранная группа заинтересованных лиц и экспертов работает совместно над определением, созданием, уточнением и достижением согласия относительно ожидаемых артефактов (таких, как модели и документы), представляющих пользовательские требования». Семинары — это специально организованные встречи со многими заинтересованными лицами и формальными ролями, такими как организатор и секретарь. В семинарах обычно участвует несколько типов заинтересованных лиц — от пользователей до разработчиков и тестировщиков. Они применяются для сбора требований одновременно у нескольких заинтересованных лиц. В группе удается эффективнее разрешать разногласия, чем при индивидуальном разговоре. Также семинары полезны, когда выявить требования требуется быстро из-за плотного графика.

Согласно одному из источников, «Организация какого-либо мероприятия — искусство управления людьми в ходе этого мероприятия, которое позволяет достичь согласованных решений в атмосфере сотрудничества, высокопроизводительного труда и заинтересованности в результатах своей работы» (Sibbet, 1994). Ответственный за мероприятие играет ключевую роль; именно он планирует семинар, отбирает участников и следит, чтобы обсуждение проводилось продуктивно. Если вы собираетесь применять новые технологии сбора требований, ответственным за первый семинар следует назначить стороннего сотрудника или второго бизнес-аналитика. В этом случае основной бизнес-аналитик сможет полностью сосредоточиться на обсуждении. Пригласите также секретаря, чтобы он фиксировал все идеи, возникающие в ходе обсуждения. Если за семинар отвечает один бизнес-аналитик, он должен четко понимать, когда играет роль ответственного и когда рядового участника обсуждения. Пригласите также секретаря, чтобы он фиксировал все идеи, возникающие в ходе обсуждения. Очень сложно одновременно вести семинар, фиксировать идеи на бумаге и принимать активное участие в обсуждении.

Семинары могут требовать много ресурсов — иногда приходится сразу собирать многих участников на несколько дней. Семинары нужно тщательно планировать, чтобы не тратить время попусту. Сводите пустую трату времени к минимуму, приходя на семинар с заранее подготовленными предварительными версиями материалов. Например, можно набросать вчерне варианты использования, которые группа проверит в процессе семинара, вместо того, чтобы формулировать их с нуля. Очень редко бывают ситуации, когда имеет смысл начинать семинар с чистого листа. Воспользуйтесь другими методами выявления требований, и лишь затем используйте семинар только для того, чтобы собрать вместе заинтересованных лиц для проработки только сложных и неясных мест.

Общие приемы координации применимы и к выявлению требований (Schwarz, 2002). О проведении семинаров для выявления требований подробно рассказано в труде Эллен Готтендинер (Ellen Gottesdiener, 2002) «Requirements by Collaboration» («Выявление требований через сотрудничество»). Она описывает массу приемов и средств для облегчения семинаров. Вот некоторые из приемов проведения эффективных семинаров выявления требований, многие из которых применимы к интервью:

Определите и отслеживайте выполнение основных правил Участники должны договориться об основных правилах проведения семинаров (Gottesdiener, 2002), своевременно начинать и заканчивать семинар, не опаздывать после перерывов, отключить звук у всех электронных приборов, не проводить несколько обсуждений одновременно, следить, чтобы каждый принимал участие в работе и комментировать и критиковать решение, а не личность. После определения правил следите, чтобы участники придерживались их.

Обеспечьте наличие всех ролей в команде Организатор должен обеспечить выполнение участниками семинара следующих задач: ведение протокола, отслеживание времени, управление базовыми правилами, а также чтобы каждый был услышан. В частности, секретарь может фиксировать на бумаге все происходящее, а кто-то другой должен следить за часами.

Планируйте повестку дня У каждого семинара должен быть четкий план, как говорится в разделе «Подготовка к выявлению требований» далее в этой главе. Заранее составьте план и повестку дня семинара и доведите их до участников, чтобы они знали, какие задачи будут обсуждаться и что ожидать и к чему готовиться.

Придерживайтесь границ проекта Чтобы удостовериться, что предлагаемые пользовательские требования не выходят за текущие границы проекта, используйте документ о концепции и границах проекта. Следите, чтоб на каждой встрече уровень обобщения соответствовал выбранным целям. Периодически участники могут углубляться в обсуждения несущественных деталей. На это уходит масса времени, которое на начальном этапе работы следует потратить на прояснение пользовательских требований — время деталей наступит позже. Задача организатора — по мере необходимости возвращать участников к теме обсуждения.

> **Внимание!** Не допускайте в процессе выявления требований обсуждения посторонних тем, например подробностей дизайна. Следите за тем, чтобы участники не отвлекались от задач встречи, убеждая их, что у них будут возможности в будущем разобраться со всеми возникающими сложностями.

Фиксируйте темы для дальнейшего обсуждения На семинаре всплывает масса случайных, но важных сведений: атрибуты качества, бизнес-правила, идеи по разработке пользовательского интерфейса, ограничения и т.п. Запишите их на плакатах — простейшим способом, — так вы не потеряете их и продемонстрируете уважение участнику, высказавшему их. Не отвлекайтесь на обсуждение деталей, не относящихся к теме дискуссии, если только они не окажутся критически важными, например бизнес-правилом, которое ограничивает варианты использования. После семинара опишите, что случилось с высказанными идеями.

Ограничивайте некоторые дискуссии по времени Подумайте о выделении фиксированного времени на обсуждение каждой темы. Возможно, некоторые дискуссии придется завершить позднее, но жесткие временные рамки помогают не тратить времени больше, чем запланировано, на первую тему обсуждения, в результате чего может не хватить времени для обсуждения остальных тем. Перед завершением обсуждения темы резюмируйте ее состояние и последующие шаги.

Не увеличивайте размер команды и тщательно отбирайте участников Небольшие группы работают намного быстрее. Семинары, число активных участников которых превышает пять или шесть человек, могут забуксовать, вылиться в параллельные дискуссии и даже ссоры. Попробуйте одновременного проводить несколько семинаров — это позволит исследовать требования различных классов пользователей. В обсуждении должны участвовать сторонник продукта и другие представители пользователей, возможно, эксперт в данной предметной области, бизнес-аналитик и разработчик. Допуском к участию в семинарах по сбору информации являются знания, опыт и право принимать решения.

У семи нянек...

Работа семинаров по сбору информации о требованиях со слишком большим количеством участников может продвигаться слишком медленно. Так, моя коллега Дебби, занимавшаяся организацией семинара, где разбирались варианты использования веб-сайта, была расстроена тем, как медленно продвигается обсуждение. Двенадцать участников длинно обсуждали детали и не могли договориться, как должен выглядеть тот или иной вариант использования. Однако темпы значительно выросли, когда Дебби сократила группу до шести человек, в число которых входили аналитик, клиент, системный архитектор, разработчик и дизайнер. Объем обсуждаемой информации стал меньше, но темпы работы более чем компенсировали эту потерю. Участникам семинара следует по его окончании обменяться информацией с коллегами, не приглашенными на дискуссию, и обсудить высказанные идеи на следующей встрече.

Вовлекайте в обсуждение каждого Иногда участники самоустраняются от обсуждения, расстроившись по какой-то причине. Возможно, их идеи не воспринимают серьезно, поскольку другим участникам их проблемы кажутся неинтересными или они не хотят отвлекать группу от текущего обсуждения. Возможно, аутсайдер не уверен в себе и уступил право голоса более активным сотрудникам или главному аналитику. Организатору семинара необходимо следить за языком телодвижений, чтобы разобраться в причинах замкнутости того или иного участника, и попытаться снова вовлечь его в работу. Видимые подсказки недоступны, если обсуждение ведется в режиме телеконференции — в этом случае вы должны внимательно слушать и ориентироваться по тону голоса. Молчаливых участников можно спросить прямо, нет ли у них мыслей, которыми они хотели бы поделиться. Организатор обязан обеспечить, чтобы все были услышаны.

Фокус-группы

Фокус-группа — это представительная группа пользователей, которые собираются в рамках управляемого мероприятия по выявлению требований для предоставления информации и идей на счет функциональных требований и требований к качеству. Встречи фокус-групп должны быть интерактивными, чтобы пользователи могли озвучить свои мысли. Фокус-группы полезны для изучения отношения, впечатления, предпочтений и потребностей пользователей (IIBA, 2009). Они особенно ценны при разработке серийных продуктов и трудности доступа к конечным пользователям.

Когда случается конфликт…

Различия во взглядах, приоритетах и личности могут приводить к конфликтам и даже гневу в группе. В таких случаях нужно немедленно устранять подобные явления. Понаблюдайте за невербальными знаками конфликта и гнева и попытайтесь понять причину. Если в группе понимают причину конфликта, у вас может появиться возможность найти решение (если оно нужно).

Если конфликтный участник отказывается работать продуктивно, поговорите с ним лично, чтобы понять, не станет ли его присутствие препятствием на пути движения группы. Если это так, можно поблагодарить участника за потраченное время и продолжить работу без него. Иногда это невозможно, и вам придется просто отказаться от встречи или вообще отказаться от обсуждения проблемной темы. Управление конфликтами — сложное искусство, которому посвящено много литературы (Fisher, Ury и Patton, 2011; Patterson et al., 2011).

Однажды я запланировал встречу, чтобы собрать бизнес-требования у нового директора по продажам. Он славился своим непростым характером, поэтому я пришел на встречу готовым внимательно слушать и понимать его желания. С первой же минуты он стал орать на меня и спраши-

вать, зачем вообще нам эта встреча. Он сказал: «Кто ты такой, чтобы думать, что имеешь право спрашивать меня о моих бизнес-целях?» Я сделал глубокий вдох и взял длинную паузу. После этого я попытался объяснить, почему мне нужно понимать его бизнес-цели: без них команде придется гадать, что нужно разработать, чтобы удовлетворить потребности клиентов, и он будет жестоко разочарован результатами. И он успокоился так же быстро, как завелся с самого начала. Без колебаний он стал активно излагать свои бизнес-цели. К счастью со мной был секретарь, который быстро фиксировал его мысли, потому что я все еще был слегка не в форме из-за неординарного приема.

Часто у вас будет большая и разнообразная база пользователей — в этом случае надо тщательно отбирать членов фокус-групп. Включите пользователей, работавших с предыдущими версиями или продуктом, похожим на тот, который планируется реализовать. Выберите пул пользователей одного типа (и создайте несколько фокус-групп для различных классов пользователей) либо отберите пул, представляющий полный спектр классов пользователей, чтобы все были представлены в равной мере.

Фокус-группы надо организовывать и контролировать. Нужно следить, чтобы участники придерживались темы, но оказывать влияния на их мнения. Можно записывать встречу, чтобы можно было вернуться и внимательно выслушать комментарии. Не стоит ожидать от фокус-групп количественного анализа — вы получите массу субъективных мнений, которые в дальнейшем можно оценивать и распределять по приоритетам в процессе разработки требований. На встречах по выявлению требований в фокус-группах можно использовать те же советы, которые были даны для семинаров. У участников фокус-групп обычно нет полномочий принимать решения по требованиям.

Наблюдение

Если вы попросите пользователей описать, как они выполняют свою работу, им наверняка будет тяжело быть точным — детали могут отсутствовать или быть некорректными. Часто это происходит из-за того, что задачи сложные и каждую мелочь не упомнишь. В других случаях причина в том, что пользователи довели выполнение задачи данных до такого автоматизма, что не состоянии сформулировать, что они в точности делают. Задача становится такой привычной, что они даже не думают о ней. Иногда можно узнать очень многое, наблюдая за тем, как пользователи в реальности выполняют свои задачи.

Наблюдения занимают много времени, поэтому они не годятся для каждого пользователя или задачи. Чтобы не нарушать ежедневную работу пользователей, ограничьте наблюдение двумя часами или меньше. Выбирайте для наблюдения важные или высокорискованные задания и множественные классы пользователей. При наблюдении в проектах гибкой разработки просите пользователей демонстрировать только задачи, относящиеся к будущей итерации.

Наблюдение рабочего процесса пользователя в рабочей среде позволяет бизнес-аналитику проверять информацию, полученную из других источников, определять новые темы для интервью, обнаруживать проблемы с текущей системой и определять возможности улучшения, чтобы новая система лучше поддерживала рабочий процесс. Бизнес-аналитик должен абстрагироваться и обобщать наблюдаемые операции пользователей, чтобы зафиксированные требования относились к классу пользователей в целом, а не к отдельным личностям. Опытный бизнес-аналитик часто может предложить идею по улучшению текущих бизнес-процессов.

Смотрите, как надо печь кекс

Чтобы продемонстрировать мощь наблюдения, расскажите нескольким своим друзьям, как испечь кекс. Вы наверняка не забудете перечислить такие шаги, как: разогреть духовку, взять нужную посуду и инструменты, добавить ингредиенты, перемешать их, вылить жидкое тесто в форму, испечь и вытащить готовый кекс из духовки. Но рассказывая о добавлении ингредиентов, не забыли ли вы сказать, что нужно открыть пакет с разрыхлителем? Или что яйца надо разбить и использовать содержимое, а скорлупу выкинуть? Эти очевидные операции могут быть не столь очевидны человеку, который никогда не занимался выпечкой.

Наблюдение может быть пассивным (молчаливым) или интерактивным. Пассивное наблюдение уместно, когда занятых пользователей нельзя отвлекать от работы. В процессе интерактивного наблюдения бизнес-аналитик может отвлекать пользователя и задавать вопросы. Это позволяет моментально понять, почему пользователь сделал именно такой выбор, или спросить, о чем он думал, предпринимая именно такое, а не иное действие. Документируйте свои наблюдения, чтобы их можно было потом проанализировать. Если позволяют корпоративные политики, может иметь смысл вести видеозапись, чтобы можно было освежить память в последующем.

Я разрабатывал приложение контактного центра для сотрудников по обслуживанию клиентов, которым нужно было просматривать печатный каталог и находить товары, которые заказывали клиенты. Команда бизнес-аналитиков встретилась с несколькими такими сотрудниками, чтобы выявить варианты использования нового приложения. Все как один сотрудники говорили о том, как сложно листать большое количество каталогов в поиске именно того товара, о котором говорил клиент. Каждый бизнес-аналитик сел рядом с одним из сотрудников и понаблюдал, как они принимают заказы по телефону. Мы увидели, что им сложно: сначала надо найти каталог по дате, а потом попытаться найти нужный товар. Наблюдения помогли нам понять, какие функции могут потребоваться в программном каталоге товаров.

Опросные листы

Опросные листы — один из способов обследования больших групп пользователей с целью выяснения их потребностей. Это недорого, хорошо подходит для сбора информации в больших сообществах пользователей и их легко использовать, даже если пользователи далеко разнесены географически. Результаты анализа опросных листов можно рассматривать, как подготовку к применению других методов выявления требований. Например, опросные листы можно применять для выявления наболевших проблем пользователей в существующей системе, а результаты использовать при обсуждении приоритетов с ответственными за принятие решений на семинаре. Опросные листы можно также использовать для получения откликов пользователей серийного продукта.

Самое сложное в опросных листах — задать правильные вопросы. Существует много рекомендаций по написанию опросных листов (Colorado State University, 2013), а мы перечислим самые важные:

- варианты ответов должны охватывать весь диапазон возможных ответов;
- варианты ответов должны быть взаимоисключающими (никаких перекрытий в числовых диапазонах) и исчерпывающими (приведите все возможные варианты и/или добавьте свободное место, куда можно вписать вариант, о котором вы не догадались);
- вопрос не должен подразумевать получение «правильного» ответа;
- шкалы величин нужно использовать единообразно во всем опросном листе;
- используйте закрытые вопросы с двумя или большим числом вариантов, если опросный лист предназначен для статистического анализа. Открытые вопросы позволяют пользователям отвечать так, как им хочется, что затрудняет поиск общих моментов в результатах;
- стоит проконсультироваться со специалистом по составлению материалов и проведению опросов, чтобы удостовериться, что вы задаете правильные вопросы «правильным» людям;
- всегда тестируйте опросный лист до его распространения. Очень обидно слишком поздно обнаружить, что вопрос был задан неоднозначно или какой-то вопрос вообще забыли добавить;
- не задавайте слишком много вопросов, потому что люди просто не станут отвечать.

Анализ системных интерфейсов

Анализ интерфейсов — независимый метод выявления требований, который подразумевает анализ систем, с которыми взаимодействует ваша система. Анализ системных интерфейсов выявляет функциональные требования к сервисам и обмену данными между системами (IIBA, 2009). Контекстные диаграммы и карты экосистем (см. главу 5) — очевидный выбор для поиска интерфейса для последующего более глубокого анализа. По сути, диаграммы

неполны, если есть интерфейс, с которым связаны требования, но его самого на диаграммах *нет*.

Для каждой системы, которая взаимодействует с вашей, определите функциональность этой системы, которая может создавать требования к вашей системе. Эти требования могут описывать, какие данные передавать или получать из другой системы и каковы правила работы с этими данными, например критерии соответствия. Может также обнаружиться существующая функциональность, которую *не нужно* реализовывать в вашей системе. Представьте, что вам нужно реализовать правила проверки заказа в корзине интернет-магазина до начала оплаты. Проанализировав системные интерфейсы, вы сможете обнаружить, что в систему управления заказами поступают заказы из других систем, которые сами их проверяют, поэтому такую функциональность реализовывать не нужно.

Анализ пользовательского интерфейса

Анализ пользовательского интерфейса — независимый метод анализа существующих систем с целью выявления пользовательских функциональных требований. Лучше всего взаимодействовать с существующими системами непосредственно, но при необходимости можно использовать снимки экрана. Руководства пользователя коробочных продуктов часто содержат снимки экрана, которые можно использовать в качестве отправного пункта. Если существующей системы нет, можно посмотреть на пользовательский интерфейс похожих продуктов.

При работе с коробочными решениями или существующей системой анализ пользовательского интерфейса помогает получить полный список экранных форм для обнаружения возможных функций. Работая с существующим пользовательским интерфейсом, можно определить стандартные операции, выполняемые пользователями в системе, и создать предварительные версии вариантов использования для проверки пользователями. Анализ пользовательского интерфейса иногда обнаруживает данные, которые должны видеть пользователи. Это хороший способ изучения, как работает существующая система (если только для этого не требуется серьезного обучения). Вместо того, чтобы спрашивать пользователей, как они взаимодействуют с системой и какие операции они выполняют, может оказаться, что вы сами можете получить начальное представление о системе.

Не стоит предполагать, что та или иная функциональность необходима в новой системе просто потому, что вы нашли ее в имеющейся системе. Кроме того, не стоит считать, что если текущая система выглядит и ведет себя определенным образом, то будущая система должна работать так же.

Анализ документов

Этот анализ подразумевает изучение всей имеющейся документации на предмет обнаружения возможных требований к ПО. К самой полезной до-

кументации следует отнести спецификации требований, бизнес-процессы, информацию о практическом опыте и руководства пользователя для существующих или аналогичных приложений. В документах могут описываться корпоративные или отраслевые стандарты, которым нужно следовать, или требования законодательства, которым должен удовлетворят продукт. При замене существующей системы в старой документации может обнаружиться как функциональность, которую нужно сохранить, так и та, от которой надо избавиться. При внедрении коробочных решений в документации поставщика рассказывается о функциональности, которая может понадобиться пользователям, но может потребоваться внимательно проанализировать, как ее можно реализовать в целевой среде. Сравнительные обзоры, которые указывают на недостатки разных продуктов, можно использовать для устранения таких недостатков в своем продукте и получения конкурентного преимущества. Отчеты о проблемах и запросы на улучшение системы, собранные отделом сопровождения у пользователей, могут содержать полезные идеи по улучшению системы в будущих выпусках.

Анализ документов — хороший способ быстро освоить текущую систему или новую предметную область. Выполнение исследования и подготовка предварительных требований способствует сокращению времени, необходимого для проведения встреч по выявлению требований. Анализ документов может выявить информацию, о которой люди не говорят вам — потому что не подумали об этом или просто не владеют ею. Например, при создании нового приложения контактного центра можно обнаружить сложную бизнес-логику, описанную в документации к текущему приложению. Пользователи могут даже не знать об этой логике. Результаты такого анализа можно использовать в качестве входных данных для интервью.

В этом методе есть риск — документы могут быть устаревшими. Требования могли измениться, а спецификации могли быть не обновлены, или в документах может описываться функциональность, которая в новой системе не нужна.

Планирование выявления требований в проекте

На ранних этапах проекта бизнес-аналитик должен создать план выявления требований. Даже простой план повышает шансы на успех и делает более реалистичными ожидания всех заинтересованных лиц. Только четко сформулировав потребности, касающиеся ресурсов, графика и выпуска, можно избежать того, что лиц, участвующих в выявлении требований, отзовут для исправления ошибок или выполнения другой работы. План выявления требований включает: приемы, которые планируется применять, когда применять эти приемы и для какой цели. Как и с любым другим планом, используйте его как руководство и письменное напоминание на протяжении проекта, но

помните, что в ходе проекта может потребоваться поменять этот план. План должен отражать:

- **Цели выявления требований** Запланируйте цели выявления требований для всего проекта и цели для каждого запланированного действия по выявлению требований.
- **Стратегии и способы выявления требований** Решите, какие приемы применять в разных группах заинтересованных лиц, например, сочетание опросных листов, семинаров, встреч с клиентами, отдельных интервью и других приемов в зависимости от доступа к заинтересованным лицам, ограничений времени и вашего знания существующей системы.
- **График и смета ресурсов** Определите, кто из разработчиков и клиентов будет участвовать в различных операциях по выявлению требований, а также примерно оцените усилия и время на выявление требований. Возможно, сначала вы сможете определить только классы, а не конкретных пользователей, но это позволит менеджерам начать планировать выделение ресурсов. Оцените время бизнес-аналитика, в том числе время на подготовку к выявлению требований и выполнение последующего анализа.
- **Документы и системы, необходимые для независимого выявления требований** Проводя анализ документов, системных интерфейсов и пользовательского интерфейса, определите материалы, которые должны быть под рукой, когда потребуются.
- **Ожидаемые результаты выявления требований** Создание перечня вариантов использования, подробная спецификация требований к программному обеспечению, анализ результатов опросных листов или спецификация атрибутов качества и производительности позволяют гарантировать, что вы выбрали для выявления требований правильных заинтересованных лиц и темы.
- **Риски, связанные с выявлением требований** Укажите факторы, которые могут помешать выполнению запланированных действий по выявлению требований, оцените серьезность каждого риска и решите, как смягчить его или управлять им. Подробнее об управлении рисками см. главу 32. Подробнее о симптомах, основных причинах и возможных решениях стандартных проблем с выявлением требований см. Приложение Б.

У многих бизнес-аналитиков есть любимые методы выявления требований — обычно это интервью и семинары, и они даже не задумываются о применении других методов, которые могут сократить потребность в ресурсах или повысить качество собираемой информации. Редко когда бывает так, что оптимальные результаты можно получить, только используя один метод выявления требований в проекте. Одни и те же методы выявления требований могут применяться в разных проектах разработки. Выбор метода зависит от особенностей проекта.

На рис. 7-3 приводятся методы выявления требований, которые чаще всего применяют в проектах того или иного типа. Выберите строку или строки,

характеризующие ваш проект и посмотрите в соответствующих столбцах, какие методы выявления требований (выделены крестиком) скорее всего подойдут в вашем случае. В частности, если вы разрабатываете новое приложение, вы скорее всего получите лучшие результаты, сочетая интервьюирование заинтересованных лиц, семинары и анализ системных интерфейсов. В большинстве проектов будут нелишними интервью и семинары. Для ПО массового рынка лучше подходят семинары для фокус-группы, потому что имеется большая база пользователей, но ограниченный доступ к их представителям. Надо понимать, что это всего лишь предложения. В частности, вы можете решить, что в проекте ПО для массового рынка нужно применить анализ пользовательского интерфейса.

	Интервью	Семинары	Фокус-группы	Наблюдение	Опросные листы	Анализ системных интерфейсов	Анализ пользовательского интерфейса	Анализ документов
ПО для массового рынка	x		x		x			
Внутрикорпоративное ПО	x	x	x	x		x		x
Замена существующей системы	x	x		x		x	x	x
Обновление существующей системы	x	x				x	x	x
Новое приложение	x	x				x		
Реализация коробочного ПО	x	x		x		x		x
Встроенные системы	x	x				x		x
Географически распределенные заинтересованные лица	x	x			x			

Рис. 7-3. Возможные методы выявления требований в проектах с разными характеристиками

Подготовка к выявлению требований

Организованные встречи по выявлению требований нужно готовить заранее, чтобы максимально эффективно использовать время всех участников. Чем больше группа, участвующая во встрече, тем важнее предварительная подготовка. На рис. 7-4 показаны действия, которые надо подготовить к встрече по выявлению требований.

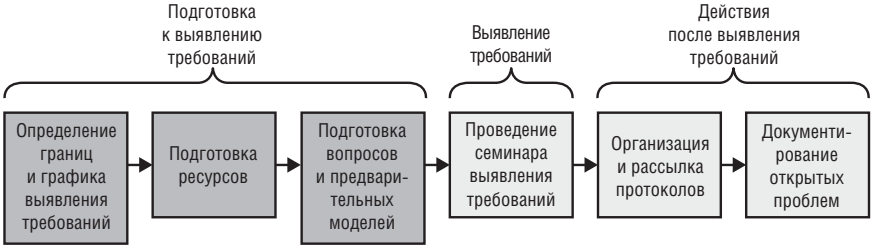


Рис. 7-4. Действия, к которым надо приготовиться перед встречей по выявлению требований

При подготовке к каждой встрече определите границы встречи, повестку дня, подготовьте вопросы и составьте предварительные версии материала, который может потребоваться на встрече. Вот несколько советов по подготовке к выявлению требований.

Определение границ и графика выявления требований Определите границы встречи по выявлению требований, учитывая необходимое время. Границы встречи можно определить в виде списка тем или вопросов, или же можно выбрать определенные потоки процессов или варианты использования, которые нужно обсудить. Определяйте границы встречи на основе границ проекта, определенных в бизнес-требованиях, чтобы при обсуждении не отвлекаться от темы. Повестка дня должна содержать темы к обсуждению, время, выделенное на каждую тему, и цели, которые планируется достичь. Заранее поделитесь планом встречи с заинтересованными лицами.

Подготовьте ресурсы Запланируйте необходимые физические ресурсы, такие как комнаты совещаний, проектов, номера телеконференций и оборудование видеоконференций. Также запланируйте время участников, не забыв о различии временных поясов, если участники находятся в географически разнесенных местах. В географически распределенных группах каждый раз изменяйте время встречи, чтобы не дискриминировать часть участников, для которых это время неудобное. Соберите документацию из разных источников. Получите доступ к необходимым системам. Пройдите онлайновый курс, чтобы изучить имеющиеся системы.

Узнайте больше о заинтересованных лицах Определите заинтересованных лиц, которые должны участвовать во встрече (см. главу 6). Узнайте о культурных и региональных предпочтениях заинтересованных лиц относительно встречи. Если для некоторых участников язык встречи неродной, стоит подумать о предоставлении им вспомогательных материалов, таких как слайды, заранее, чтобы у них было время подготовиться и возможность следить за ходом встречи. Слайды могут содержать конкретные вопросы, которые вы планируете задавать, или просто предоставлять контекст встречи, который вы будете раскрывать устно. Постарайтесь избежать противопоставления «нас» и «их».

Подготовьте вопросы Начинать встречу всегда нужно со списком заранее подготовленных вопросов. Используйте области неопределенности в предварительных моделях (которые описываются в следующем разделе) как источник вопросов. Если вы готовитесь к интервью или семинару, используйте другие методы выявления требований для формулировки неразрешенных вопросов. Существует много источников возможных вопросов для выявления требований (Wiegers, 2006; Miller, 2009).

Формулируйте вопросы так, чтобы они не вели в тупик или не подводили к определенному ответу. Как аналитик вы должны копнуть поглубже требования, представленные клиентами, чтобы понять их настоящие потребности. Если пользователям задать вопрос: «Что вы хотите?», вы получите массу случайной информации, оставляющей аналитика в замешательстве. Вопрос: «Что

вам требуется делать?» — гораздо лучше. Задав несколько раз вопрос «почему», можно перенести обсуждение из плоскости представленного решения в плоскость четкого понимания проблемы, которую надо решить. Задавайте открытые вопросы, чтобы понять текущие бизнес-процессы пользователя и понять, как новая система может повысить их производительность.

Вообразите, что вы обучаетесь служебным обязанностям пользователя или непосредственно выполняете их под его руководством. Какие задачи вам придется решить? Какие вопросы у вас возникают? Другой подход — влезть в «шкуру» новичка, которого обучает опытный пользователь. Вы задаете массу вопросов, а он управляет беседой, выбирая важную, с его точки зрения, тему для обсуждения.

Попробуйте метод исключений. Что мешает пользователю успешно выполнить задачу? Как система должна реагировать на различные ошибочные условия? Задавайте вопросы, начинающиеся со слов: «А что еще могло бы...», «Что произойдет, когда...», «Вам когда-нибудь требовались...», «Где вы получаете...», «Почему вы делаете (или не делаете)...» и «А кто-нибудь когда-либо...». Задокументируйте источник каждого требования, чтобы, если потребуется, понять их причину и проследить, на каких же потребностях клиентов основываются те или иные направления разработки.

Как и при любой модификации, неудовлетворение настоящим дает отличную пищу для принятия нового или улучшенного будущего состояния. Когда вы разрабатываете приложение, призванное заменить текущую унаследованную версию системы, попросите пользователей назвать три вещи, которые раздражают их сейчас в системе больше всего. Этот вопрос поможет вам понять, чего же ждут пользователи от новой системы.

Вам не нужен, и не обязателен идеальный список вопросов перед началом интервью или семинара. Подготовленные вопросы помогут, если разговор застопорится. Вопросы должны выглядеть естественно и удобно — как разговор, а не опрос. Несколько минут спустя после начала встречи вы можете обнаружить, что упустили важную часть обсуждения. Будьте готовы отложить свои вопросы, если это потребуется. В конце встречи поинтересуйтесь: «Есть ли что-либо еще, о чем я должен был спросить?», чтобы попытаться вытащить на «белый свет» вопросы, о которых вы просто не подумали.

Подготовьте предварительные модели Во время встреч по выявлению требований могут использоваться модели анализа, чтобы пользователи предоставляли более качественные требования. Одни из самых полезных моделей — варианты использования и потоки процессов, потому что они тесно связаны с тем, как люди представляют себе свою работу. Создавайте предварительные модели до начала встреч по выявлению требований. Предварительная модель служит исходным пунктом, который позволяет вам освоить тему, а пользователей заставляет думать об идеях. Скорректировать предварительную модель проще, чем создавать новую с нуля.

Если вы новичок в предметной области, вам будет сложно самостоятельно создать предварительную модель. Используйте другие методы выявления

требований, чтобы набраться достаточно знаний, с которыми можно уже что-то делать. Читайте существующие документы, анализируйте существующие системы на предмет моделей, которые можно повторно использовать в качестве исходной точки, или проведите интервью в формате «один на один» с экспертом предметной области, чтобы получить информацию, с которой можно начать. После этого сообщите группе, с которой работаете: «Скорее всего эта модель неправильная. Порвите ее на лоскуты и покажите мне, как она должна выглядеть. Я не обижусь».

Выявление требований

На рис. 7-5 показаны действия, которые надо выполнить на встрече по выявлению требований.

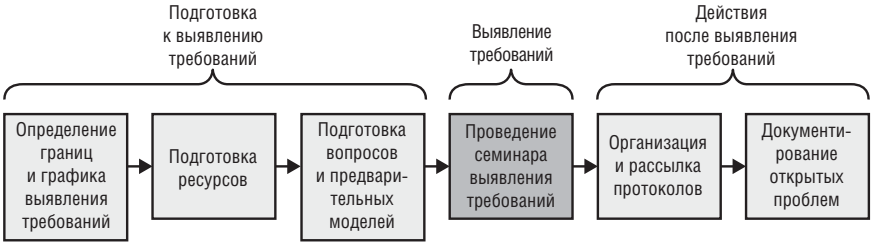


Рис. 7-5. Выполнение действий по выявлению требований на одной отдельно взятой встрече

Само по себе выявление требований сравнительно очевидно: если это интервью, вы разговариваете с людьми, если это анализ документов, вы читаете документы. Но при организации мероприятия по выявлению требований могут быть полезными следующие советы.

Объясняйте заинтересованным лицам свои методы Разъясните заинтересованным лицам особенности своего подхода к выявлению требований и почему вы применяете именно такой подход. Объясните методы исследования, которые вы собираетесь использовать, такие как варианты использования или потоки процессов, а также как они помогут заинтересованным лицам предоставить более качественные требования. Также опишите, как вы будете фиксировать предоставляемую ими информацию и отправлять им материалы на проверку после встречи.

Обеспечьте качественное протоколирование Назначьте секретарем человека, который не будет активно участвовать в дискуссии, но который будет отвечать за точное ведение протокола встречи. Протокол встречи должен содержать список присутствующих и отсутствующих участников, принятые решения, действия, которые нужно предпринять, имена ответственных за выполнение этих действий, оставшиеся открытыми вопросы и ключевые моменты обсуждения. К сожалению, бизнес-аналитики иногда ведут встречи без выделенного секретаря, что заставляет их брать на себя эту роль. В такой ситуации будьте готовы писать скорописью, быстро печатать или использовать записывающее устройство (с согласия участников). Специализированные

устройства, называемые аудио-ручками могут превращать сделанные вручную записи в электронную форму и связывать их с аудиозаписью обсуждения. Можно также использовать доску или бумагу на стене и фотографировать сделанные на ней заметки.

Подготовьте вопросы заранее, чтобы долго не думать на месте для поддержания обсуждения. Используйте скоропись, чтобы записать вопрос, пришедший на ум во время монолога одного из участников, чтобы при возможности можно было бы быстро вернуться к нему. Не пытайтесь зафиксировать диаграммы с помощью сложной программы построения диаграмм — просто сфотографируйте наброски диаграмм или быстро перерисуйте от руки.

Задействуйте физическое пространство В большинстве комнат четыре стены — задействуйте их во время встречи для рисования диаграмм и списков. Если на стенах нет досок для рисования, закрепите на стенах больших листы бумаги. Также держите под рукой наклейки для заметок и маркеры. Предложите другим участникам вставать с места и тоже делать заметки — такая организация позволяет вовлечь людей в процесс. Готтенсдинер (Gottesdiener, 2002) называет этот прием совместной работы «стеной чудес». Если у вас есть готовые артефакты, на которые можно посмотреть, (такие как предварительные модели, существующие требования или системы), спроектируйте их на стену.

Организация совместных встреч с географически распределенными участниками требует больше изобретательности. Можно использовать средства организации конференций для обмена слайдами и взаимодействия с участниками. Если несколько участников находятся в одной комнате, используйте средства видеоконференций, чтобы показать удаленным участникам рисунки на стенах и досках.

Активные участники

Однажды я проводил семинар по выявлению требований на заводе по производству полупроводников с десятком инженеров. Я использовал доску, на которой рисовал технологические процессы по мере продвижения разговора. Каждый раз по завершении процесса я останавливался и фотографировал доску, прежде чем перейти к следующему процессу. В середине дня первой встречи один из инженеров спросил, не позволю ли я ему порисовать на доске. Я с удовольствием передал маркер. Он уже освоил нотацию диаграмм, а поскольку он был экспертом по системе, то смог легко нарисовать технологический процесс на доске. После этого он прошелся по процессу, уточняя или проверяя информацию у коллег. Он взял руководство процессом на себя, что позволило мне сосредоточиться на зондирующих вопросах и ведении заметок. Вскоре все инженеры включились в процесс, и маркер пошел по рукам, так что у каждого была возможность высказаться.

Если обычаи позволяют, можете использовать игрушки, чтобы стимулировать мыслительные процессы участников или дайте им что-нибудь, чтобы

им было чем занять свои руки. Простые игрушки могут способствовать рождению идей. В одной команде проводили мозговой штурм по определению бизнес-целей проекта. Вначале дня я дал каждому участнику пластилин и попросил с его помощью смоделировать концепцию своего продукта — никаких дополнительных инструкций я не дал. Я разбудил их фантазию и заставил их думать творчески, и они получили от этого удовольствие. Мы перенесли эту энергию в написание реальной концепции продукта.

Действия после выявления требований

По завершении мероприятий по выявлению требований все еще остается много работы. Нужно организовать и разослать протоколы, задокументировать открытые проблемы и классифицировать собранную информацию. На рис. 7-6 показаны действия, которые выполняются после встречи по выявлению требований.

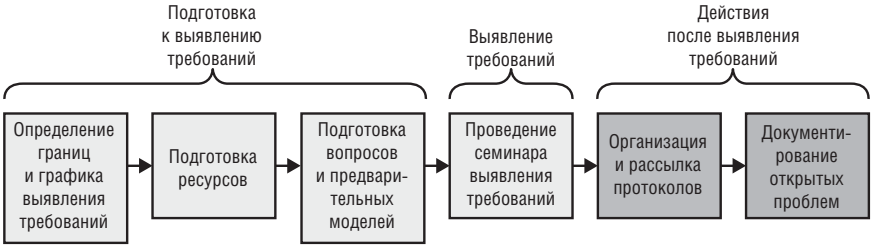


Рис. 7-6. Действия после проведения встречи по выявлению требований

Организация и рассылка протоколов

Если вы провели интервью или семинар, организация заметок и протоколов скорее всего потребует больше усилий, чем при организации информации в процессе независимого выявления требований. Консолидируйте информацию, полученную из многих источников. Пересматривайте и обновляйте свои заметки сразу после завершения встречи, пока информация еще свежа в памяти.

Редактирование протоколов выявления требований представляет определенный риск. Вы можете некорректно вспомнить, что означал определенный предмет и, сами не желая того, изменить его значение. Храните набор исходных заметок, чтобы при необходимости к ним можно было вернуться. После каждого интервью или семинара рассылайте консолидированные заметки и протоколы участникам с просьбой проверить их, чтобы убедиться, что они правильно отражают результаты встречи. Просмотр на ранних стадиях необходим для успешного сбора требований, поскольку только те люди, которые эти требования предоставили, могут судить, правильно ли они зафиксированы. Проводите дополнительные обсуждения, если это позволит разрешить все противоречия и заполнить пробелы. Подумайте, может имеет смысл по-

делиться сводными заметками и протоколами с заинтересованными лицами проекта, которые не участвовали во встрече, чтобы они были в курсе происходящего. Это позволит им оперативно указать на какие-либо проблемы и сложности.

Документирование открытых проблем

Во время мероприятий по выявлению требований вам будут встречаться вещи, нуждающиеся в исследовании в дальнейшем, или пробелы в знаниях, которые нужно закрыть. Или при анализе своих заметок у вас могут возникать новые вопросы. Проанализируйте материалы встреч по выявлению требований на предмет оставшихся открытыми проблем и зафиксируйте их в средстве отслеживания дефектов. Для каждого вопроса запишите все имеющие к нему отношение заметки, уже достигнутые результаты, ответственного за вопрос и срок разрешения. Есть смысл использовать то же средство отслеживания дефектов, что используют команды разработки и тестирования.

Классификация предоставляемой клиентом информации

Не следует ожидать, что ваши клиенты представят краткий, полный и хорошо организованный список своих потребностей. Аналитикам придется классифицировать массу полученных данных о требованиях, чтобы их удалось задокументировать и использовать соответствующим образом. На рис. 7-7 показано девять таких категорий требований. Во время выявления требований делайте быстрые заметки, если обнаружите информацию, относящуюся к указанным типам. Например, запишите «ТД» в кружке, если речь идет о требованиях к данным.

Рис. 7-7. Классификация предоставляемой клиентом информации

Как и при любой другой классификации, часть информации может не попасть точно ни в одну из этих категорий, например:

- требования, не относящиеся к разработке ПО, например необходимость обучения пользователей работе с новой системой;
- ограничение проекта, например затраты или ограничения, налагаемые графиком (в отличие от ограничений, касающихся дизайна или реализации);
- предположение или зависимость;
- дополнительная информация хронологического или описательного характера, а также относящаяся к управлению контекстом;
- избыточная информация, не несущая дополнительной ценности.

Участники выявления требований не скажут вам: «Вот бизнес-требование». Как аналитик вы сами должны определить, к какому типу информации относится каждое услышанное вами утверждение. Далее приводятся примеры фраз, за которыми нужно следить в процессе классификации.

Бизнес-требования Вся информация, описывающая финансовые, рыночные или другие отношения коммерческого характера, которые клиенты или компания-разработчик собираются получить от использования продукта, относится к бизнес-требованиям (см. главу 5). Обращайте внимание в разговоре на такие высказывания о преимуществах, которые покупатели или пользователи ПО хотят получить:

- «Увеличить рыночную долю в регионе с X% до Y% в течение следующих Z месяцев».
- «Сэкономить Y долларов в год за счет сокращения использования электроэнергии, которая сейчас расходуется неэффективно».

Пользовательские требования Основные утверждения пользователей о преследуемых ими целях или бизнес-задачах представляют собой пользовательские требования, чаще всего представленными как варианты использования, сценарии или пользовательские истории (см. главу 8). Сотрудник, который говорит: «Мне нужно сделать то-то и то-то», вероятнее всего описывает конкретный вариант использования, например:

- «Мне нужно напечатать почтовую наклейку на посылку»;
- «Как ведущему оператору мне необходимо каждое утро калибровать контроллер насоса».

Бизнес-правила Если клиент заявляет, что только определенные классы пользователей могут выполнять определенные действия при определенных условиях, он, возможно, описывает бизнес-правило (см. главу 9). Вообще-то это не требования к ПО, но с них можно вывести функциональные требования, чтобы обеспечить выполнение этих правил. Такие фразы, как «Должно соответствовать...», «Если <условие верно>, тогда <должно произойти то-то>» или «Должно вычисляться в соответствии с...», подразумевают, что пользователь описывает бизнес-правило. Вот несколько примеров.

- «Новый клиент должен авансом оплатить 30% гонорара за консультацию и дорожные расходы».
- «Одобрение отпусков должно соответствовать политике отпусков отдела управления персоналом».

Функциональные требования Функциональные требования описывают ожидаемое поведение системы при определенных условиях и действия, которые система позволит выполнять пользователям. Ниже перечислены примеры функциональных требований, которые вы можете услышать от пользователей:

- «Если давление превышает 40,0 фунтов на квадратный дюйм, должен загореться предупредительный световой сигнал»;
- «У пользователя должна быть возможность сортировать список проектов в прямом и обратном алфавитном порядке».

Эти высказывания демонстрируют, как пользователи обычно представляют функциональные требования, однако их нельзя их записать непосредственно в спецификацию требований к программному обеспечению. Бизнес-аналитик должен превратить эти высказывания в более точные спецификации. Подробнее о написании качественных функциональных требований см. главу 11.

Атрибуты качества Утверждения, насколько хорошо система выполнять что-то, называются атрибутами качества (см. главу 14). Обращайте внимание на слова, которыми пользователи описывают желаемые характеристики системы: быстрая, легкая, интуитивно понятная, удобная для пользователя, устойчивая к сбоям, надежная и эффективная. Вам придется поработать с пользователями, чтобы понять, что именно они имеют в виду под этими неясными и субъективными терминами, и зафиксировать четкие, поддающиеся проверке атрибуты качества. Вот примеры атрибутов качества, выраженные словами пользователей:

- «Мобильное ПО должны быстро реагировать на сенсорные команды».
- «Механизм корзины покупок должен быть простым в использовании, чтобы новые посетители не отказывались от покупки».

Требования к внешнему интерфейсу Требования в этой категории описывают связь вашей системы с остальным миром. Шаблон спецификации требований к ПО в главе 10 содержит разделы, касающиеся взаимодействия с пользователями, оборудованием и другими программными системами. Такие фразы, как «Должна распознавать сигналы от...», «Должна отправлять сообщения в адрес...», «Должна быть в состоянии читать (или записывать) файлы в формате <*формат*>» и «Элементы пользовательского интерфейса должны соответствовать стандарту <*стандарт*>», указывают, что клиент описывает требования к внешнему интерфейсу. Вот несколько примеров:

- «Производственная система должна управлять сортировкой кремниевых пластин».

- «Мобильное приложение должно отсылать образ чека в банк, после фотографирования размещаемого чека».

Ограничения Ограничения, касающиеся дизайна и реализации, официально ограничивают возможности, доступные разработчику (см. главу 14). Для устройств со встроенным ПО часто необходимо учитывать физические ограничения, такие, как размер, вес и соединения в плате. Вот примеры фраз, указывающих на то, что клиент описывает ограничения дизайна или реализации: «должна быть написана на определенном языке программирования», «не может превышать определенного значения» или «должна использовать определенный элемент управления пользовательского интерфейса». Вот какие ограничения может выдвинуть клиент:

- «Размер файлов, представленных в электронном виде, не должен превышать 10 МБ»;
- «В браузере для всех безопасных транзакций следует использовать 256-битное шифрование».

Как и в случае с функциональными требованиями, задача аналитика шире, чем просто запись высказываний относительно ограничений. Поинтересуйтесь причинами ограничений, проверьте, насколько они аргументированы, и задокументируйте обоснование для включения этого ограничения в качестве требования.

Определения данных Каждый раз, когда клиенты описывают формат, тип данных, допустимые значения или значение по умолчанию для элемента данных или структуры бизнес-данных или отчета, который нужно построить, они дают определения данных (см. главу 13). Вот несколько примеров требований к данным:

- «Почтовый индекс в США состоит из пяти цифр, за которым следует необязательный дефис и еще четыре цифры — по умолчанию 0000».
- «Заказ состоит из идентификационной информации клиента, адреса и подробностей доставки и одного или большего числа товаров, у каждого из которых есть номер, число единиц, цена за единицу и общая сумма».

Идеи, касающиеся решений Большинство информации, которую пользователи представляют как требования, подпадает под определение идей о решении. Те, кто описывают определенный способ взаимодействия с системой для выполнения определенного действия, представляют допустимое решение. Бизнес-аналитику необходимо научиться преодолевать поверхностные формулировки и докапываться до сути идеи, которая и представляет собой требование. Многократные вопросы, «почему» пользователю нужно, чтобы все работало именно так, скорее всего позволит обнаружить реальную потребность (Wiegers, 2006). Например, пароли — лишь одно из возможных решений требования к безопасности. Вот примеры идей решения:

- «Затем в раскрывающемся списке я выбираю штат, куда я хочу отправить посылку».

- «Телефон должен позволять пользователю перемещаться между экранами жестом смахивания».

В первом примере фрагмент *«в раскрывающемся списке»* тянет на решение, потому что описывает конкретный элемент управления на пользовательском интерфейсе. В этом случае благоразумный бизнес-аналитик задаст вопрос: «Почему в раскрывающемся списке?» Если пользователь ответит: «Мне просто кажется, так удобно», то непосредственно требование можно сформулировать примерно так: «Система должна позволять пользователю выбрать штат, куда он хочет отправить посылку». Однако пользователь может сказать: «Я предложил раскрывающийся список, потому что мы применяем этот элемент в других областях, и я хочу единообразия. Кроме того, так пользователь не сможет ввести непроверенные данные». Это отличные причины для выбора определенного решения. Однако следует помнить, что, записывая идею о решении в требование, вы тем самым налагаете ограничение по дизайну на это самое требование: реализация требования становится возможной только в одном варианте. Это не обязательно плохо или неправильно, просто убедитесь, что для ограничения есть убедительная причина.

Классификация предоставляемой клиентом информации — всего лишь начало процесса создания спецификации требований: вам нужно собрать информацию в четко определенные и организованные наборы требований. По мере работы с этой информацией нужно четко сформулировать конкретные требования и сохранить их в соответствующих разделах шаблонах документов или репозитории команды. Пройдитесь по информации несколько раз, чтобы убедиться, что все утверждения являются высококачественными требованиями, как описано в главе 11. При обработке протоколов мероприятий по выявлению требований, отмечайте части готовыми, когда сохраняете их в правильном месте.

Как понять, что сбор требований завершен

Каких-либо признаков, показывающих, что выявление требований завершено, нет. В сущности, полностью закончить не удастся никогда, особенно если система специально разрабатывается итерациями, например в проектах гибкой разработки. Когда люди размышляют по утрам в душе и разговаривают со своими коллегами, у них возникают идеи насчет добавления дополнительных или изменения существующих требований. Следующие признаки подскажут вам, что источники сведений уже почти иссякли, по крайней мере на текущем этапе:

- пользователи уже не могут придумать каких-либо еще вариантов использования или пользовательских историй. Обычно они описывают их в порядке убывания значимости последних;
- пользователи предлагают новые варианты использования, однако они не приводят к появлению новых функциональных требований. Эти «новые»

предложения могут оказаться случаями других вариантов использования, которые вы уже рассмотрели;

- пользователи повторно описывают уже обсуждавшиеся проблемы;
- предлагаемые новые функции, пользовательские или функциональные требования выходят за рамки проекта;
- вновь предлагаемые требования имеют низкий приоритет;
- пользователи предлагают возможности, которые можно реализовать «когда-то позже», а не включить «в конкретный продукт, который мы сейчас обсуждаем»;
- разработчики и тестировщики, проверяющие требования, задают очень мало вопросов.

Собрать воедино мнения многих пользователей трудно при отсутствии хорошо структурированной организационной схемы, такой, как вариант использования или разделы в спецификации требований. Несмотря на все ваши усилия собрать *все* требования, вам это не удастся, поэтому будьте готовы вносить изменения по мере продвижения. Помните, что ваша задача — накопить согласованное представление требований, которое достаточно для продолжения работы над следующим выпуском или итерацией при приемлемом уровне риска.

Несколько советов о том, как собирать информацию

Мастерство управления дискуссией по выявлению требований приходит с опытом, очень помогают в таком деле тренинги, где слушатели учатся брать интервью, общаться в группах, разрешать конфликтные ситуации и т. п. Но несколько советов позволят сократить время обучения.

Баланс представления заинтересованных лиц Проблемы возникают, если вам приходится выслушивать нескольких пользователей, и если вы имеете дело с наиболее громогласным и упрямым клиентом. Вы можете упустить из виду требования, важные для определенных классов пользователей, или включить требования, которые не отражают потребности большинства. Чтобы соблюсти баланс, следует привлечь несколько горячих сторонников продукта, обладающих полномочиями для выступления от лица соответствующих классов пользователей, причем каждого из них должны поддерживать несколько представителей этого же класса пользователей.

Правильное определение границ проекта В процессе выявления требований иногда выясняется, что границы проекта определены неправильно — они либо слишком широкие, либо слишком узкие. В первом случае вам придется собрать дополнительные требования, которые позволят принести пользу бизнесу и клиенту, при этом процесс неизбежно затянется. Во втором случае высказанные пользователями потребности безусловно важны, но выходят за границы текущего проекта. Это означает, что определенные

границы, возможно, слишком малы для получения удовлетворительных результатов. Таким образом, сбор пользовательских требований иногда влечет за собой изменение концепции или границ проекта.

Избегайте конкуренции между требованиями и дизайном Часто говорят, что суть требований заключается в том, *что* система должны делать, а то, *как* решение будет реализовано, относится к области дизайна. Хотя эта формулировка заманчиво кратка, по сути она примитивна. Сбор информации по информации должен быть сосредоточен на этом «*что*», но области анализа и дизайна разделяет размытая линия, а не четкая граница (Wiegers, 2006). Гипотетические «*как*» помогают уточнить и детализировать понимание потребностей пользователей. Модели анализа, наброски экранных форм и прототипы помогают в ходе сбора информации более осязаемо выразить потребности и избежать ошибок и упущений. Объясните пользователям, что эти средства — только для иллюстрации идей и не обязательно попадают в окончательные решения.

Исследование в разумных пределах Увлечение исследованиями иногда приводит к проблемам с выявлением требований. Высказана прекрасная идея или предложение, но для ее оценки необходима обстоятельная проверка. Выяснение ее осуществимости или ценности следует воспринимать как самостоятельную задачу внутри проекта. И в этом случае можно использовать прототипы. Если для вашего проекта необходимо провести обширные исследования, применяйте поэтапную разработку, чтобы исследовать требования постепенно, небольшими и безопасными частями.

Подразумеваемые и неявные требования

Вы никогда не сможете задокументировать все сто процентов требований к системе. Но требования, которые вы не указали, действительно представляют риск того, что в проекте будет создано решение, отличающееся от того, что ожидают заинтересованные лица. Два вероятных виновника нереализованных ожиданий — подразумеваемые и неявные требования.

- *Подразумеваемые требования* (assumed requirements) — то, что люди ожидают получить, явно не выражая требования. Что-то очевидное для вас может быть не таким очевидным для разработчиков.
- *Неявные требования* (implied requirements) — те, что необходимы по причине другого требования, но явно не сформулированы. Разработчики неспособны реализовывать функциональность, о которой они не знают.

Для снижения подобных рисков старайтесь выявить пробелы в знаниях, которые, как ожидается, будут восполнены подразумеваемыми и неявными требованиями. Во время встреч по выявлению требований скрашивайте: «Что вы считаете само собой разумеющимся?», чтобы обнаружить такие скрытые мысли. Натолкнувшись на подразумеваемое утверждение во время обсуждения требований, запишите его и в дальнейшем проверьте. Люди предполага-

ют, что все должно быть так, как всегда было, потому что они привыкли к существующей системе и бизнес-процессам. Если вы разрабатываете систему, которая заменит существующую, проанализируйте функции предыдущей системы, чтобы определить, действительно ли они нужны в новой системе.

Для выявления неявных требований изучите результаты начальных встреч по выявлению требований, чтобы определить области, нуждающиеся в уточнении. Может туманное высокоуровневое требование надо наполнить информацией, чтобы все заинтересованные лица хорошо понимали его? Или может у требования (скажем, у возможности сохранения не до конца заполненной веб-формы) отсутствует логическое парное требование (загрузка ранее сохраненной формы для дальнейшей работы)? В поисках отсутствующих требований может потребоваться повторно проинтервьюировать некоторых заинтересованных лиц (Rose-Coutre, 2007). Также подумайте о привлечении к выявлению таких требований новых заинтересованных лиц, которые знакомы с предметом и могут обнаружить пробелы.

Читайте между строк, чтобы определить те возможности или характеристики, которые клиенты полагают само собой разумеющимися и даже не считают нужным обрисовать. Задавайте бесконтекстные высокоуровневые и открытые вопросы, допускающие разные толкования, чтобы выявить информацию о глобальных характеристиках бизнес-проблем и их возможных решениях (Gause и Weinberg, 1989). Реакция клиентов на такие вопросы, как «Какой уровень точности необходим в продукте?» или «Почему вы не согласны с ответом Мигеля?», иногда более действенны, чем вопросы с ответами типа «да/нет» или «A/B/C».

Никаких подразумеваемых требований

Мне как-то пришлось работать с командой разработки, которая реализовывала информационный портал, который должен был выполнять много функций, в том числе загрузку, редактирование и публикацию информации на веб-сайте. В портале уже было примерно тысяча элементов информации, объединенных в иерархию. В команде управления содержимым сайта предполагали, что пользователи получат возможность быстро перемещаться по иерархии и находить нужную информацию. Члены команды не удосужились сформулировать требование к навигации пользователей по сайту. Однако когда разработчики реализовали пользовательский интерфейс для навигации по содержимому в виде одного уровня, а не иерархии, с отображением на странице 20 элементов. Были части содержимого, для поиска которого требовалось пролистать свыше 50 страниц. Если бы разработчики и члены команды потратили больше времени на диалог, можно было бы избежать значительных переделок.

Поиск упущенных требований

Пропуск каких-либо важных данных — самый распространенный недостаток требований. Обнаружить их в процессе повторного просмотра требований очень трудно, поскольку их просто не видно! Предлагаемые далее приемы позволяют выявить упущенные требования.

- Раскладывайте требования высокого уровня на простейшие составляющие — это позволит понять, чего же именно просят пользователи. Из-за неясности требований высокого уровня, предоставляющих клиентам свободу интерпретации, возможно несовпадение представлений клиента о продукте и тем, что создает разработчик.
- Убедитесь, что все классы пользователей предоставили вам информацию и что у каждого пользовательского требования есть по крайней мере один класс пользователей, который выиграет от реализации этого требования.
- Подробно документируйте, на каких функциональных требованиях основаны требования к системе, пользовательские требования списки откликов на события и бизнес-правила, — это позволит вам быть уверенным, что аналитик описал всю необходимую функциональность.
- Для выявления недостающих требований проверяйте пограничные значения. Предположим, в одном требовании указано: «Если стоимость заказа меньше 100 долларов, стоимость доставки составит 5,95 долларов», а в другом — «Если стоимость заказа превышает 100 долларов, стоимость доставки составляет 5% от общей стоимости заказа». А как быть, если стоимость заказа составляет ровно 100 долларов? Это не оговорено, значит, отсутствует соответствующее требование или как минимум неудовлетворительно задокументировано.
- Используйте разнообразные формы представления информации о требованиях. Трудно прочитать большой объем текста и заметить, что чего-то не хватает. Некоторые модели анализа визуально представляют требования высокого уровня абстракции — лес, а не отдельные деревья. Рассматривая модель, вы можете заметить, что от одного блока к другому должна идти стрелка, — это тоже недостающее требование. Подробнее о моделях анализа — в главе 12.
- Наборы требований со сложной булевой логикой (несколько операторов «И», «ИЛИ» и «НЕ») часто оказываются неполными. Если для комбинации логических условий не определено соответствующее функциональное требование, разработчику приходится догадываться, как же должна действовать система, или искать ответ на этот вопрос. Условие «Иначе» часто упускают. Чтобы убедиться, что вы рассмотрели все возможные ситуации, представляйте сложную логику с помощью таблиц и деревьев решений (подробнее — в главе 12).
- Создайте контрольный список стандартных функциональных областей, которые надо учитывать в своих проектах: ведение журнала ошибок, ар-

хивирование и восстановление, безопасность доступа, отчетность, печать, возможности предварительного просмотра и конфигурирования пользовательских параметров. Периодически сверяйте контрольный список с уже определенными функциями на предмет выявления пробелов.

- Один из способов поиска недостающих требований — создать модель данных. Все сущности данных, с которыми работает система, должны иметь соответствующую функциональность для их создания, чтения из внешних источников, обновления текущих значений и/или удаления. Часто эти четыре стандартные операции обозначают акронимом CRUD (Create, Read, Update, Delete — создание, чтение, обновление, удаление). Убедитесь, что в вашем приложении можно указать функциональность для выполнения этих операций во всех сущностях, в которых это требуется (см. главу 13).

> **Внимание!** Остерегайтесь *паралича аналитического процесса*: не тратьте слишком много времени на выявление требований, пытаясь не упустить ни одно из них.

Скорее всего вы никогда не выявите все требования к продукту, но практически у каждой команды разработчиков есть возможность улучшить процесс выявления требований за счет применения описанных в этой главе приемов.

Что дальше?

- Вспомните недостающие требования, которые были выявлены на поздних стадиях проекта. Почему они были упущены из виду в процессе сбора информации? Как вы могли бы выявить их раньше? Какую пользу это принесло бы вашей организации?
- Выберите часть любого задокументированного мнения клиента или раздела спецификации требований к программному обеспечению. Классифицируйте каждый элемент в этой области требований по категориям, показанным на рис. 7-7. Если вы обнаружите элементы, которые вы неправильно классифицировали, переместите их в соответствующую категорию в документации требований.
- Перечислите методы выявления требований, использовавшиеся вами в предыдущем и текущем проекте. Какие методы себя хорошо зарекомендовали, а какие — нет и почему? Определите способы выявления требований, которые, по-вашему, дадут лучший результат, и решите, как вы будете применять их в следующий раз. Подумайте, какие препятствия могут помешать вам применить эти способы на практике, и определите, как их преодолеть.

Глава 8

Как понять требования пользователей

Участники проекта Chemical Tracking System проводили свой первый семинар, посвященный требованиям, чтобы выяснить, как химики будут использовать новую систему. На него были приглашены бизнес-аналитик Лори, сторонник продукта от химиков Тим, еще два представителя химиков, Сэнди и Питер, а также ведущий разработчик Рави.

«Тим, Сэнди и Питер определили приблизительно 14 вариантов использования Chemical Tracking System, — сообщила группе Лори. — Вы присвоили высший приоритет варианту использования «Заказ химиката», и Тим уже составил его краткое описание, поэтому с него и начнем. Тим, как ты представляешь себе заказ химиката с помощью системы?»

«Прежде всего, — сказал Тим, — вам следует знать, что только те люди, у которых есть разрешение руководителей лабораторий, могут заказывать химикаты».

«Ладно, это похоже на бизнес-правило, — ответила Лори. — Я начну составлять список бизнес-правил, поскольку, вероятно, будут и другие. Видимо, нам придется проверять, включен ли пользователь в утвержденный список лиц». После этого она управляла обсуждением того, как участники семинара представляют себе процесс формирования заказа нового химиката. Ей понадобилось несколько листов блокнота и бумаги для заметок, чтобы свести вместе информацию о предварительных условиях, выходных условиях, а также взаимодействии пользователя и системы. Лори поинтересовалась, чем будет отличаться вариант использования, когда пользователь заказывает химикат у поставщика, от того, когда на складе. А кроме того, какие проблемы при этом могут возникнуть и как система должна обрабатывать каждую ошибку? По истечении получаса удалось выработать четкое представление о том, как пользователь будет заказывать химикат. Участники семинара были готовы обсуждать следующий вариант использования.

При проектировании ПО, которое удовлетворяет потребности пользователей, необходимо понять, что они собираются делать с его помощью. Некоторые команды используют подход, ориентированный на продукт. Основа определения функций, которые надо реализовать, программный продукт, при этом предполагают, что эти функции понравятся будущим клиентам. Но в боль-

шинстве случаев лучше при выявлении требований применять подход, ориентированный на пользователя или на работу с продуктом. Ориентация на пользователей и ожидаемые варианты использования позволяет избежать создания функций, которые никто не будет применять, а также правильно расставит приоритеты.

Пользовательские требования размещены на втором уровне требований, показанных на рис. 1-1 в главе 1. Они находятся между бизнес-требованиями, определяющими цели проекта, и функциональными требованиями, описывающими, что разработчики должны реализовать. В этой главе рассказывается о двух наиболее часто используемых способах анализа пользовательских требований: о вариантах использования и пользовательских историях.

В течение многих лет аналитики извлекали информацию о пользовательских требованиях из сценариев использования (Alexander и Maiden, 2004). Вариант на основе использования был формализован в виде подхода, в котором для моделирования требований используются варианты использования продукта (Jacobson et al., 1992; Cockburn, 2001; Kulak и Guiney, 2004). Позже приверженцы гибкой разработки (agile) ввели понятие «пользовательской истории» — краткого утверждения, выражающего потребность пользователя и служащего начальной точкой дискуссий, в процессе которых определяются детали (Cohn, 2004).

Как варианты использования, так и пользовательские истории, смещают акцент с ориентированного на продукт представления на обсуждение того, что *пользователям* нужно выполнить, а не спрашивать их, что он хотят от *системы*. Цель этого подхода в том, чтобы описать задачи, которые пользователям нужно выполнять посредством системы, или взаимодействия пользователя и системы, которые дадут полезный результат для того или иного заинтересованного лица. Такое понимание позволяет бизнес-аналитику вывести нужную функциональность, которая должна быть реализована, чтобы сделать возможным такие сценарии использования. Также при этом создаются тесты, призванные проверить, что функциональность была реализована правильно. Ориентированные на использование стратегии выявления требований дают более точное понимание требований пользователей во многих классах проектов, чем это возможно в других применяемых нами методах.

Варианты использования и пользовательские истории хорошо работают при выявлении требований для бизнес-приложений, веб-сайтов, терминалов и систем, которые предоставляют пользователям возможность управлять какими-то устройствами. Но они не подходят для понимания требований в приложениях определенных типов. У таких продуктов, как процессы пакетной обработки, системы для интенсивных вычислений и приложения для хранения данных, может оказаться всего несколько простых вариантов использования. Сложность их работы заключается в выполняемых вычислениях или генерировании отчетов, а не во взаимодействии пользователя и системы.

Варианты использования и пользовательские истории не подходят для описания встроенных и других систем реального времени. Представьте себе автоматическую мойку машин. У владельца авто есть только одна задача — помыть машину, возможно с небольшими вариациями: обработка низа кузова, нанесение защитного воска, полировка. Но на мойке машин происходит много вещей. Есть механизм перемещения машины, множественные моторы, насосы, клапана, переключатели, элементы управления и освещения, а также таймеры или сенсоры для управления этими физическими компонентами. Надо также позаботиться о диагностической функциональности, такой как уведомление оператора о том, что заканчивается моющая жидкость, а также есть требования по обнаружению ошибок и по безопасности. Что произойдет, если сломается механизм перемещения, когда автомобиль находится в мойке, или если сломается сушка? При выявлении требований для систем реального времени обычно создают список внешних событий, на которые должна реагировать система, и соответствующих вариантов ответной реакции системы. Подробнее об анализе событий см. главу 12.

Варианты использования и сценарии использования

Вариант использования (use case) описывает последовательность взаимодействия системы и внешнего действующего лица, в результате которого действующее лицо получает полезный результат. Имена вариантов использования всегда пишутся в формате «глагол + объект». Надо выбирать надежные описательные имена, из которых было бы понятно, что вариант использования обеспечит что-то ценное для какого-то пользователя. В таблице 8-1 перечислены примеры некоторых вариантов использования из разных приложений.

Табл. 8-1. Примеры вариантов использования в различных приложениях

Приложение	Примеры вариантов использования
Система отслеживания химикатов	Заказать химикат
	Распечатать спецификации безопасности материалов
	Изменить заказ химиката
	Проверить состояние заказа
	Сгенерировать квартальные отчеты об использовании химикатов
Терминалы регистрации в аэропорту	Зарегистрироваться на рейс
	Распечатать посадочные талоны
	Изменить места
	Сдать багаж
	Оплатить более комфортное место

Табл. 8-1. (*окончание*)

Приложение	Примеры вариантов использования
Бухгалтерская система	Создать счет-фактуру
	Сверить банковскую выписку
	Ввести транзакцию кредитной карты
	Распечатать налоговые декларации для поставщиков
	Найти конкретную транзакцию
Книжный интернет-магазин	Обновить профиль клиента
	Найти товар
	Купить товар
	Отследить отгруженную партию
	Отменить поставку заказа

В применении к проектам гибкой разработки пользовательская история это «короткое, простое описание функции с точки зрения человека, которому нужна эта новая возможность, обычно это пользователь или клиент системы» (Cohn, 2010). Пользовательские истории пишутся в соответствии со следующим шаблоном, хотя возможны и другие стили:

```
Как <тип пользователя>, я хочу <цель>, чтобы <причина>.
```

Такой шаблон обладает преимуществом перед более коротким названием варианта использования, потому что хотя они оба описывают цель пользователя, но пользовательская история определяет класс пользователя и обоснование запрашиваемой функциональности системы. Это очень ценные дополнения. Класс пользователя, который не обязательно должен быть человеком, в пользовательской истории соответствует основному действующему лицу в варианте использования (подробнее об этом рассказывается далее в статье). Обоснование может предоставляться в кратком описании варианта использования. Таблица 8-2 демонстрирует, как можно выразить некоторые из вариантов использования из таблицы 8-1 в виде пользовательских историй.

Табл. 8-2. Примеры вариантов использования и соответствующих пользовательских историй

Приложение	Пример варианта использования	Соответствующая пользовательская история
Система отслеживания химикатов	Заказать химикат	Как химик я хочу заказать химикат, чтобы выполнять эксперименты
Терминал регистрации в аэропорту	Зарегистрироваться на рейс	Как пассажир я хочу зарегистрироваться на рейс, чтобы улететь в место назначения
Бухгалтерская система	Создать счет-фактуру	Как владелец маленькой компании я хочу создать счет-фактуру, чтобы получить оплату от клиента

Табл. 8-2. (*окончание*)

Приложение	Пример варианта использования	Соответствующая пользовательская история
Книжный интернет-магазин	Обновить профиль клиента	Как клиент я хочу обновить свой профиль, чтобы все последующие покупки оплачивать новой кредитной картой

На таком уровне вариант использования очень похож на пользовательскую историю — оба нацелены на понимание, что различным типам пользователей нужно для выполнения определенного взаимодействия с программной системой. Однако с этих похожих начальных точек два процесса движутся в разных направлениях, как показано на рис. 8-1. Оба подхода могут давать другие результаты, такие как визуальные модели анализа, но рис. 8-1 иллюстрирует фундаментальное различие.

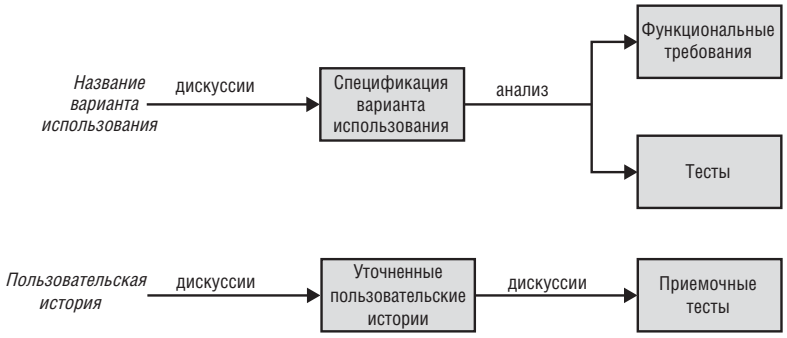


Рис. 8-1. Как пользовательские требования ведут к функциональным требованиям и тестам при использовании подхода на основе вариантов использования и подхода на основе пользовательских историй

При применении вариантов использования следующим делом бизнес-аналитик должен поработать с представителями пользователей, чтобы понять, как они представляют себе порядок диалога в системе при выполнении варианта использования. Бизнес-аналитик структурирует информацию, собранную в соответствии с шаблоном варианта использования; пример мы покажем далее в этой главе. В шаблоне много места для хранения информации, которая дает детальное представление о варианте использования и предоставляет сопутствующие сведения. Не обязательно полностью заполнять шаблон, если разработчики могут получить нужную информацию из короткой спецификации, но обращение к шаблону во время выявления требований помогает участникам обнаруживать всю необходимую информацию. На основе спецификации варианта использования бизнес-аналитик может сформулировать функциональные требования, которые разработчики должны реализовать, а тестировщики должны определить тесты, которые позволяет определить, правильно ли реализован вариант использования. Разработчики могут реализовать вариант использования целиком в течение

одного выпуска или итерации. С другой стороны, можно реализовать только часть определенного варианта использования по причинам размера или приоритета, после чего реализовать оставшиеся части в последующих выпусках.

Используемая в проектах гибкой разработки пользовательская история служит заменой будущих дискуссий, которые произойдут строго по времени между разработчиками, представителями пользователей и бизнес-аналитиком (если таковой имеется в проекте). Эти дискуссии служат для выявления дополнительной информации, которой должны обладать разработчики, чтобы реализовать пользовательскую историю. Уточнение пользовательских историй в процессе обсуждений приводит к созданию набора более мелких и четко направленных историй, которые описывают отдельные части функциональности системы. Пользовательские истории, слишком большие для реализации в одной итерации гибкой разработки [их еще называют *эпиками* (epic)], разбиваются на более мелкие истории, которые могут реализовываться в одной итерации. Подробнее об эпиках и пользовательских историях см. главу 20.

В командах гибкой разработки не определяют функциональные требования, а обычно тщательно перерабатывают пользовательскую историю в набор приемочных тестов, которые в совокупности описывают «условие удовлетворения» истории. Обдумывание тестов на таком раннем этапе — отличная идея в любых проектах, независимо от используемых в них методов разработки. Мышление в терминах тестирования помогает выявить вариации базовой пользовательской истории (или варианта использования), исключительные условия, которые надо обработать, а также нефункциональные требования, такие как соображения производительности и безопасности. Если разработчик реализовал код, достаточный для удовлетворения приемочных тестов, а значит, выполнил условие удовлетворения истории, тогда пользовательская история считается корректно реализованной.

Пользовательские истории предоставляют собой краткую формулировку потребностей пользователей. Варианты использования предоставляют более подробные сведения о том, как пользователь представляет себе взаимодействие с системой для решения задачи. В варианте использования не должно присутствовать деталей дизайна — только мысленная картинка в уме пользователя о том, как должно выглядеть взаимодействие. Пользовательские истории предоставляют удобство простоты и лаконичности, но это компромисс. Варианты использования предоставляют участникам проекта структуру и контекст, которых нет в наборе пользовательских историй. Они предоставляют бизнес-аналитику возможность упорядоченными образом вести дискуссии по выявлению требований, выходя за рамки простого составления списка вещей, которые пользователям нужно реализовать с помощью системы, и использования этого списка в качестве исходной точки для планирования и обсуждения.

Не все разделяют убеждение, что пользовательские истории являются адекватным решением для крупных и более требовательных проектов (Gilb и

Gilb, 2011). При использовании вариантов использования вы можете изучать отдельные части (направления, предварительные и выходные условия и т. п.) для выявления важных функциональных и нефункциональных требований, а также составления тестов. Это помогает не пропустить никаких требований, которые разработчики должны реализовать, чтобы пользователи могли выполнять вариант использования. Но в пользовательских историях такая структура и жесткость отсутствуют, поэтому команде проще упустить некоторые приемочные тесты. Бизнес-аналитик или разработчик должен иметь опыт разработки пользовательских историй, чтобы не упустить необходимую функциональность. Анализ вариантов использования может обнаружить, что в нескольких вариантах использования есть похожие исключения (и другие общие черты), тогда скорее всего это можно реализовать как одну единообразную стратегию обработки ошибок в приложении. В наборе пользовательских историй обнаружить такие общие черты сложнее.

Подробнее о том, как выявлять и применять пользовательские истории при изучении пользовательских требований, см. работы Конов (Cohn, 2004 и Cohn, 2010) или Леффингвелла (Leffingwell, 2011). Оставшаяся часть этой главы посвящена способу с использованием варианта использования с указанием общих черт и различий со способом на основе пользовательских историй, когда это уместно по ходу рассказа.

Способ с применением варианта использования продукта

Как уже говорилось, вариант использования описывает последовательность взаимодействий системы и внешнего действующего лица, приводящего к результату, который представляет ценность для этого лица. *Действующим лицом* (actor) может быть человек (или иногда другая программная система или аппаратное устройство), взаимодействующий с системой для реализации варианта использования. Например, в варианте использования Chemical Tracking System «Заказ химиката» участвует действующее лицо «Сотрудник, разместивший заказ на химикат». Класса пользователей Chemical Tracking System с таким именем не существует. И химики, и специалисты, работающие на складе химикатов, могут запрашивать химикаты, поэтому члены любого из этих классов могут играть роль такого сотрудника. Вот примеры вопросов, которые вы можете задать, чтобы помочь представителям пользователей определить действующих лиц:

- Кто (или что) уведомляется, если что-то происходит внутри системы?
- Кто (или что) предоставляет системе информацию и сервисы?
- Кто (или что) помогает системе среагировать и выполнить задачу?

Пользователи и действующие лица

Различие между пользователями и действующими лицами может быть не всегда очевидным (Wiegers, 2006). Представьте себе, что у пользователя-человека есть набор табличек с именами действующих лиц, которых система распознает и позволяет участвовать во вполне определенных вариантах использования. Когда пользователь хочет выполнить определенное действие в системе, он надевает соответствующую табличку. Система распознает этого человека как действующее лицо, указанное на табличке, и запускает вариант использования, в котором оно заинтересовано. Когда сотрудник хочет заказать химикат, он «надевает» табличку «Сотрудник, разместивший заказ на химикат» и система Chemical Tracking System считает его именно таковыми, независимо от его должности. То есть в тот момент пользователь играет роль «Сотрудник, разместивший заказ на химикат». Сотрудник склада тоже может надевать эту табличку. Как у химика, так и сотрудника склада есть набор других табличек с именами других действующих лиц, известных системе Chemical Tracking System. Ну, вообще-то, на самом деле у них нет всех возможных табличек, но так удобнее объяснять. Пользователи — это реальные люди (или системы); действующие лица — это абстракции.

Диаграммы вариантов использования (use-case diagrams) позволяют получить высокоуровневое визуальное представление о требованиях пользователей. На рис. 8-2 показан фрагмент диаграммы варианта использования Chemical Tracking System, где применяется нотация UML (Unified Modeling Language — унифицированный язык моделирования) (Booch, Rumbaugh, and Jacobson, 1999; Podeswa, 2010). Прямоугольник показывает границы системы. Линии соединяют каждое действующее лицо (человечек) с вариантами использования (эллипсами), с которыми это лицо взаимодействует. Стрелка от действующего лица к варианту использования указывает, что это *основное действующее лицо* (primary actor) этого вариант использования. Основное действующее лицо инициирует вариант использования и получает основную пользу от него. Стрелка идет от варианта использования к дополнительному действующему лицу (secondary actor), которое каким-то образом участвует в успешном выполнении варианта использования. В качестве дополнительных действующих лиц часто выступают другие программные системы, в фоновом режиме вносящие свой вклад в выполнение варианта использования. Таким дополнительным действующим лицом выступает база данных по обучению на рис. 8-2. Эта система задействуется, если сотрудник запрашивает опасный химикат, для работы с которым требуется пройти курс по обращению с опасными материалами.

Обратите внимание на сходство этой диаграммы варианта использования с контекстной диаграммой на рис. 5-6. Оба определяют границу между объектами, находящимися за пределами системы, и вещами внутри системы.

В диаграмме варианта использования прямоугольник отделяет некоторые внутренние элементы высокого уровня системы — варианты использования — от внешних действующих лиц. Контекстная диаграмма также отображает объекты, расположенные вне системы, но на ней не видны внутренние части системы. Стрелки в контекстной диаграмме указывают на поток данных, сигналы управления или физические материалы (если вы определили, что «система» должна включать ручные процессы), пересекающие границу системы. С другой стороны, стрелки диаграммы вариантов использования просто указывают на связи между действующими лицами и вариантами использования, в которых они участвуют, а не на какие-либо потоки. Как и во всех других формах представления требований, все читатели создаваемых вами моделей должны понимать используемую вами нотацию.

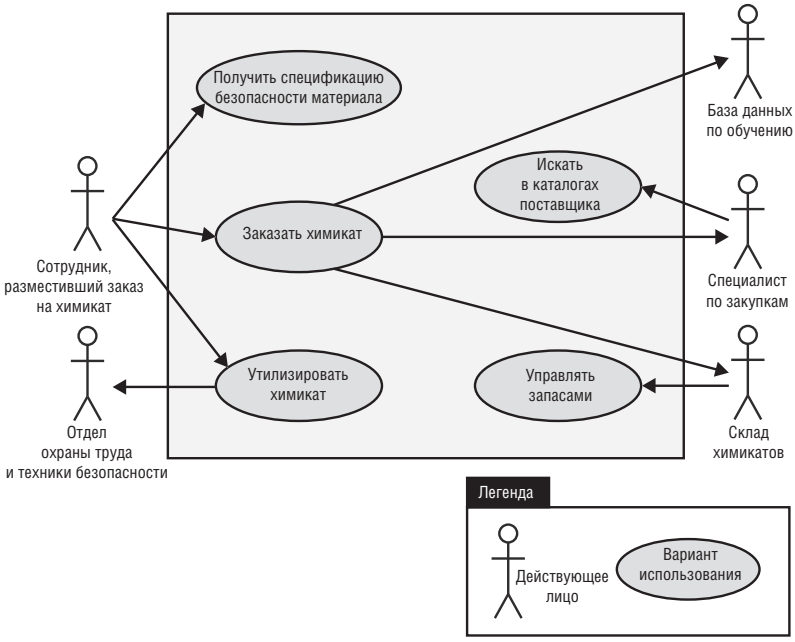


Рис. 8-2. Фрагмент диаграммы варианта использования системы Chemical Tracking System

Варианты использования и сценарии использования

Вариант использования — это отдельное, независимое действие, которое действующее лицо может выполнить для получения определенного значимого результата. Один вариант использования может охватывать несколько схожих действий с одной целью. *Сценарий* — это описание одного случая использования системы. Следовательно, он представляет собой набор связанных между собой сценариев использования, где сценарий — это отдельный пример варианта использования. Вы можете начать разработку пользовательских требований с общего определения варианта использования, а затем

на его основе создать конкретные сценарии использования или, наоборот, перейти от некоего сценария к более широкому варианту использования.

На рис. 8-3 показан полный шаблон варианта использования, заполненный в качестве примера данными, взятыми из системы Chemical Tracking System. В Приложении В приведено еще несколько вариантов использования, созданных на основе этого шаблона. Как и другие шаблоны, его не нужно заполнять сверху вниз, и вся информация шаблона нужна далеко не в каждом варианте использования. Шаблоны всего лишь предоставляют структуру, в которой надо хранить информацию, обнаруженную в процессе обсуждения варианта использования, в упорядоченном и единообразном виде. Шаблон напоминает обо всей той информации, которую надо учесть при рассмотрении варианта использования. Если информация, которая должна быть в шаблоне, уже есть в других местах, просто включите ее по ссылке. Например, не включайте текст каждого бизнес-правила, которое влияет на вариант использования в шаблоне, а просто укажите идентификаторы соответствующих бизнес-правил, чтобы при необходимости читатель мог найти эту информацию.

Идентификатор и название:	UC-4 Запросить химикат
Автор:	Лори **Дата создания:** 22.08.13
Основное действующее лицо:	Сотрудник, разместивший заказ на химикат **Дополнительное действующее лицо:** Покупатель; Склад химикатов; База данных по обучению
Описание:	Сотрудник, разместивший заказ на химикат, указывает в запросе необходимый химикат, вводя его название или идентификатор или импортируя его структуру из соответствующего графического средства. Система выполняет запрос, предлагая контейнер с химикатом со склада или позволяя создать запрос на заказ у поставщика.
Триггер:	Сотрудник указывает, что хочет заказать химикат.
Предварительные условия:	PRE-1. Личность пользователя аутентифицирована. PRE-2. Пользователь имеет право запрашивать химикаты. PRE-3. База данных по запасам химикатов в данный момент доступна.
Выходные условия:	POST-1. Запрос сохраняется в Chemical Tracking System. POST-2. Запрос отправлен на склад химикатов или поставщику.
Нормальное направление развития варианта использования:	**4.0 Запросить химикат со склада** 1. Сотрудник указывает требуемый химикат. 2. Система перечисляет контейнеры с необходимым химикатом, имеющиеся на складе. 3. Сотрудник может просмотреть историю любого контейнера. 4. Сотрудник выбирает определенный контейнер или просит отправить запрос поставщику (см. 4.1). 5. Сотрудник вводит остальную информацию, чтобы завершить запрос. 6. Система сохраняет запрос и отправляет его на склад химикатов.
Альтернативное направление развития варианта использования:	**4.1 Запросить химикат у поставщика** 1. Сотрудник ищет химикат по каталогам поставщика (см. 4.1.E1). 2. Система отображает список поставщиков, где также указаны размеры, класс и цена контейнеров. 3. Сотрудник выбирает поставщика, размер, класс и количество контейнеров. 4. Сотрудник вводит остальную информацию, необходимую для запроса. 5. Система сохраняет запрос и перенаправляет его поставщику.
Исключения:	**4.1.E1. Химиката нет в продаже** 1. Система отображает сообщение «У поставщиков нет такого химиката». 2. Система предлагает сотруднику запросить другой химикат или выйти из программы. 3а. Сотрудник просит запросить другой химикат. 4а. Система заново начинает нормальное направление варианта использования. 3б. Сотрудник решает выйти из системы. 4б. Система завершает вариант использования.
Приоритет:	Высокий
Частота использования:	Примерно пять раз в неделю каждым химиком, 200 раз в неделю каждым работником склада.
Бизнес-правила:	BR-28, BR-31
Специальные требования:	Система должна импортировать химические структуры в стандартной закодированной форме из любых средств, поддерживающих рисование химических структур.
Предположения:	Импортированные химические структуры должны быть верными.

Рис. 8-3. Фрагмент описания варианта использования «Запросить химикат»

К необходимым элементам описания варианта использования относятся:

- уникальный идентификатор и краткое имя, определяющее цель пользователя;
- краткое текстовое описание цели варианта использования;
- условие-триггер, инициирующее выполнение варианта использования;
- список предварительных условий (ноль или больше), которые должны быть удовлетворены до начала выполнения варианта использования;
- одно или больше выходных условий, описывающих состояние системы после успешного завершения варианта использования;
- пронумерованный список действий, иллюстрирующий последовательность этапов взаимодействия действующего лица и системы (диалогов) от предварительных до выходных условий.

Соглашение о метках в варианте использования

Спецификации вариантов использования состоят из большого числа небольших пакетов информации: нормального и альтернативного направлений, исключений, предварительных и выходных условий и т. п. Пример на рис. 8-3 иллюстрирует простое соглашение о метках, которые дает четкое представление о происходящем. В каждом варианте использования применяется последовательность чисел и содержательное название, отражающее задачу пользователя: UC-4 «Запросить химикат». Идентификатор нормального направления в данном варианте использования — «4.0». Альтернативные направления идентифицируются по числу справа, так что номер первой альтернативы — «4.1», второй — «4.2» и т. д. Как у нормального, так и у альтернативных направлений есть свои исключения. Первое исключение в нормальном направлении варианта использования «4» обозначается как «4.0.E1». Второе исключение в первом альтернативном направлении — «4.1.E2».

Предварительные и выходные условия

Предварительные условия (preconditions) определяют, что должно присутствовать до выполнения варианта использования в системе. Система должна иметь возможность проверить все предварительные условия, чтобы определить, возможно ли выполнять вариант использования. Предварительные условия могут описывать состояние системы (для варианта использования, заключающегося в выдаче денег из банкомата, в нем должны быть деньги), но не намерения пользователя («Мне нужно немного денег»).

При обнаружении события-триггера, информирующего, что пользователь хочет выполнить определенный вариант использования, система говорит себе (и не обязательно пользователю): «Секундочку, мне нужно проверить наличие предварительных условий». Само по себе событие-триггер не является одним из предварительных условий. Система начинает выполнение варианта использования только при выполнении всех предварительных усло-

вий. Проверка предварительных условий обычно предотвращает возникновение некоторых ошибок, которые возникали бы из-за того, что система «знает» об отсутствии всех или части этих условий, но все равно вынуждена пытаться выполнить вариант использования. Если банкомат пустой, он не должен позволят пользователю начинать операцию снятия денег. Это позволяет сделать приложения более надежными. Пользователи могут не знать обо всех предварительных условиях варианта использования, поэтому бизнес-аналитику может потребоваться получить соответствующую информацию из других источников.

Выходные условия (postconditions) описывают состояние системы после успешного выполнения варианта использования. Выходные условия могут описывать:

- что-то, что пользователь может наблюдать сам (система отображает остаток на счете);
- физические результаты (банкомат выдал деньги и распечатал чек);
- изменение внутреннего состояния системы (счет дебетован на сумму выданных средств и применимых комиссий).

Многие выходные условия очевидны для пользователя, потому что отражают полезный для него результат: «Я получил свои наличные!» Но никакой пользователь никогда не скажет бизнес-аналитику, что система должна скорректировать информацию о наличных средствах в банкомате путем уменьшения баланса на только что выданную сумму. Пользователи не знают, да и не хотят знать о таких внутренних подробностях работы системы. Но разработчики и тестировщики должны знать о них, а это означает, что бизнес-аналитик должен выявить такие условия, например в процессе общения со специалистом предметной области, и зафиксировать их в дополнительных выходных условиях.

Нормальные и альтернативные направления и исключения

Один сценарий считается нормальным направлением развития (normal flow) варианта использования, его также называют основным направлением, базовым направлением, нормальным направлением, основным сценарием, главным успешным сценарием и благоприятным путем. Нормальное направление для варианта использования «Заказ химиката» — заказ химиката, который есть на складе. Как показано на рис. 8-3, нормальное направление представлено номерованным списком операций, указывающих конкретные сущности — систему или конкретное действующее лицо, выполняющие эти операции.

Другие успешные сценарии из варианта использования, называются *альтернативными потоками* (alternative flows) или *вторичными сценариями* (secondary scenarios) (Schneider и Winters, 1998). Они также могут привести к успешному выполнению задания и удовлетворяют выходным условиям варианта использования, но они представляют менее популярные или менее приоритетные вариации самой задачи или способа ее выполнения.

В определенной точке принятия решений в диалоговой последовательности нормальное направление может перейти в альтернативное, а затем вернуться (или нет) обратно в нормальное. Пользователь, говорящий «По умолчанию должно происходить...», описывает нормальное направление варианта использования. Такое утверждение, как «У пользователя также должна быть возможность заказать химикат у поставщика», предлагает альтернативное направление, показанное на рис. 8-3 под номером «4.1» и ответвляющееся от этапа «4» нормального направления.

Вспомните, что пользовательские истории представляют собой лаконичные формулировки их потребностей, чем истории отличаются от более подробных описаний в вариантах использования. В мире гибкой разработки пользовательская история иногда «закрывает» такую же область, что и вариант использования, но обычно пользовательские истории представляют лишь один сценарий или альтернативное направление. При обсуждении требований к системе Chemical Tracking System в команде гибкой разработки могут придти к таким пользовательским историям:

- Как химик я хочу заказать химикат, чтобы выполнять эксперименты.
- Как химик я хочу заказать химикат со склада, чтобы получить его немедленно.
- Как химик я хочу заказать химикат у поставщика, потому что не доверяю чистоте веществ, имеющихся на складе.

Первая из этих трех историй соответствует варианту использования в целом. Вторая и третья истории соответственно представляют основное и первое альтернативное направление (см. рис. 8-3).

Условия, препятствующие успешному выполнению варианта использования, называются *исключениями* (exceptions). Исключения описывают ожидаемое ошибочное условие, которое может сложиться во время выполнения варианта использования, и как его обрабатывать. В некоторых случаях пользователь может устранить исключение, например путем повторного ввода данных, которые были неправильными. Но в других ситуациях вариант выполнения должен завершиться безуспешно. Для варианта использования «Запросить химикат» есть исключение «Химиката нет в продаже» — оно отмечено как 4.1.E1 на рис. 8-3. Если в процессе сбора информации вы не укажете, как обрабатывать исключение, то возможны два пути:

- разработчики будут стараться угадать, как поступать при возникновении исключений, что ведет к разнородной обработке ошибок и сниженной надежности приложения;
- когда пользователь натолкнется на ошибку, произойдет сбой системы, так как никто не предусмотрел такой ситуации.

Можно быть уверенным, что сбои системы не входят в список требований пользователей.

Некоторые ошибки могут влиять на несколько вариантов использования или операций в нормальном направлении одного варианта использования.

В качестве примеров можно привести разрыв сетевого подключения, отказ базы данных в середине операции или физический сбой устройства, например замятие бумаги. Такие ситуации нужно рассматривать как дополнительные функциональные требования, которые надо реализовать, а не считать их исключениями во всех затронутых вариантах использования. Задача заключается в том, чтобы не размещать принудительно всю функциональность в одном варианте использования. Нужно применять ориентированное на использование выявление требований, чтобы определить столько жизненно важной функциональности системы, сколько возможно.

Не обязательно реализовывать все альтернативные направления, определенные для варианта использования — часть можно отложить на последующие итерации или выпуски. Тем не менее, вы *должны* реализовать исключения, которые не позволяют успешно выполнить те направления, что вы реализуете. Опытные программисты знают, что обработка исключений требует много усилий по кодированию. Пропущенные исключения часто становятся причиной нереализованных требований. Определение условий исключений во время сбора требований помогает командам создавать надежные продукты. Этапы нормального направления указывают, где могут произойти известные исключения, указывая на раздел шаблона варианта использования, где описывается, как обрабатывать это исключение.

В проектах гибкой разработки, в которых используются пользовательские истории, проблема исключений решается путем создания приемочных тестов, создаваемых для каждой истории. Третья из приведенных выше пользовательских историй предусматривает заказ химиката у поставщика. В процессе обсуждения этой истории могут возникнуть вопросы, например: «Что если нужного химиката нет ни у одного поставщика?» В результате этого можно сформулировать такой приемочный тест: «Если нужного химиката нет в имеющихся каталогах поставщиков, отобразить соответствующее сообщение». Как и в любом другом подходе к тестированию, набор приемочных тестов пользовательской истории должен покрывать как ожидаемое поведение, так и случаи неправильного использования системы.

Хотя большинство вариантов использования можно описать простым языком, блок-схема или UML-диаграмма позволяют визуально представить логическое развитие сложного варианта использования, как показано на рис. 8-4. Блок-схема и диаграммы процесса показывают точки принятия решений и условия, при которых основное направление развития событий переходит в альтернативное.

В примере на рис. 8-3 конечная цель действующего лица одна и та же — запрос химиката, поэтому оба сценария входят в один вариант использования. Некоторые этапы альтернативного направления аналогичны этапам нормального направления, но для завершения альтернативного направления необходимо выполнить уникальные действия. В нашем случае альтернативное направление может позволить пользователю выполнить поиск необхо-

димого химиката по каталогам поставщиков, после чего можно вернуться в нормальное направление и продолжить процесс заказа с этапа 4.

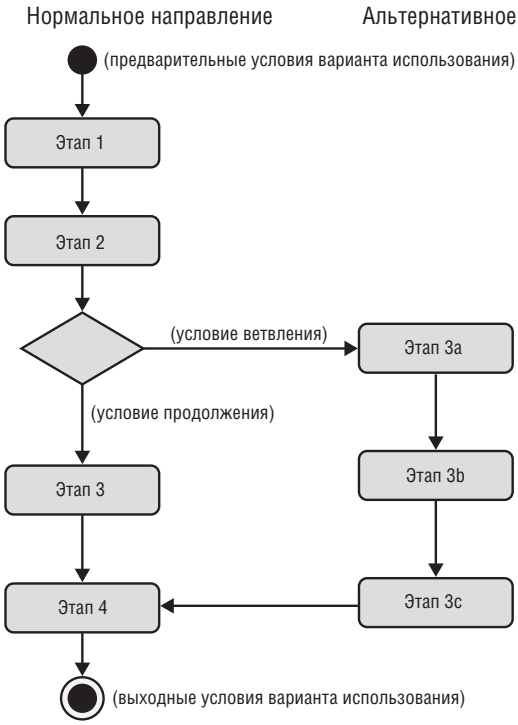


Рис. 8-4. Диаграмма, иллюстрирующая диалоговый поток при нормальном и альтернативном развитии варианта использования

Формат вариантов использования

Исчерпывающая спецификация варианта использования требуется не всегда. Кокберн (Cockburn, 2001) пишет о шаблонах *свободного* и *полноформатного* варианта использования. Сводный вариант использования представляет собой простое текстовое представление задачи пользователя и взаимодействий с системой, например только раздел «Описание» из рис. 8-3. Весь шаблон на рис. 8-3 иллюстрирует полноформатный вариант использования. И, конечно же, возможны любые промежуточные варианты. И не обязательно все свои варианты использования документировать с одинаковым уровнем детализации. Иногда для описания подлежащей реализации функциональности достаточно названия варианта использования и короткого описания. В других случаях можно просто перечислить альтернативные направления и исключения, не детализируя их. Но иногда, в случае сложных вариантов использования, команде нужна более обширная спецификация. Полноформатные варианты использования полезны, когда:

- представители пользователей не могут тесно взаимодействовать с командой разработки на протяжении проекта;
- приложение сложное, а сбои системы создают высокий риск;
- варианты использования представляют собой совершенно новые требования, с которыми разработчики никогда не сталкивались;
- варианты использования являются самыми детальными требованиями, которые смогут получить разработчики;
- на основе пользовательских требований планируется создать исчерпывающие варианты тестирования;
- распределенным тесно взаимодействующим командам требуется подробный совместно используемый массив информации.

Не нужно быть слишком догматичным в отношении объема информации, которая должна присутствовать в варианте использования: помните, что ваша задача — понять задачи пользователя настолько, чтобы разработчики могли начать работу над системой при минимальном риске возможных переделок.

Расширение и включение

В диаграмме вариантов использования между вариантами возможны отношения одного из двух типов: расширение (extend) и включение (include). На рис. 8-3 показано, что нормальное направление варианта использования «Заказ химиката» заключается в заказе химиката на складе, а альтернативное направление — заказ у поставщика. На диаграмме варианта использования на рис. 8-2, у сотрудника, разместившего заказ на химикат, есть вариант использования «Искать в каталогах поставщика». Представьте, что нужно позволить этому сотруднику выполнять вариант использования «Искать в каталогах поставщика» как вариант при заказе химиката в рамках альтернативного направления обработки. В диаграмме варианта использования может отображаться, что отдельный вариант использования, такой как «Искать в каталогах поставщика» *расширяет* нормальное направление в альтернативное направление, как показано на рис. 8-5 (Armour и Miller, 2001).

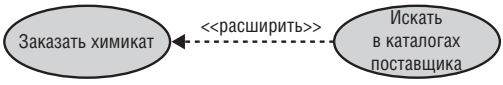


Рис. 8-5. Пример варианта использования включает взаимосвязь «расширить» в системе Chemical Tracking System

Иногда несколько вариантов использования имеют общие наборы операций. Чтобы избежать повторения этих операций в каждом варианте использования, определите отдельный вариант с общей функциональностью и укажите, что он *включен* в другие варианты использования как подвариант. Эта процедура аналогична вызову общей подпрограммы в компьютерной программе. В качестве примера рассмотрим работу бухгалтерской программы. Возможны два варианта использования — «Оплатить счет» и «Подтвердить

кредитную карту», причем в обоих, чтобы выполнить платеж, пользователь должен подписать чек. Вы можете создать отдельный вариант использования «Выписать чек», который содержит набор действий, необходимых для выписки чека. Этот вариант будет включен в обе транзакции, как показано на рис. 8-3. «Подписать чек» является отдельным вариантом использования, потому что это одна из задач, которую можно выполнять с помощью этой бухгалтерской программы.

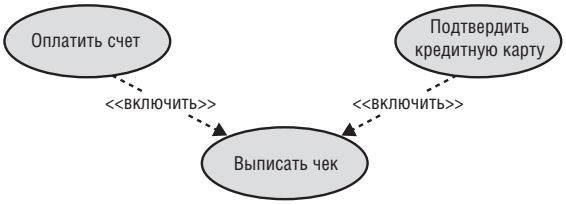


Рис. 8-6. Пример варианта использования включает взаимосвязь «включить» в бухгалтерском приложении

> **Внимание!** Не затягивайте обсуждение того, когда, как и стоит ли вообще использовать взаимоотношения типа «расширить» и «включить». Один автор книг о вариантах использования сказал мне, что такие взаимоотношения лучше всего обсуждать с друзьями за бокалом пива.

Соответствие предварительных и выходных условий

Для многих систем пользователь может связать последовательность вариантов использование в «макровариант использования», описывающий объемную задачу. Для интернет-магазина предлагаются, например, такие варианты использования: «Искать по каталогу», «Добавить товар в корзину» и «Оплатить товары в корзине». Если вы можете выполнить все эти действия по отдельности, то все они считаются независимыми вариантами использования. То есть на веб-сайте у вас может быть один сеанс, в котором только что выполнен поиск по каталогу, второй сеанс, в котором в корзину только что добавлен товар без предварительного поиска (например, путем ввода номера товара), и третий сеанс, в котором выполнена оплата товаров в корзине (это подразумевает, что ваша корзина должна сохраняться на протяжении сеансов входа). Кроме того, вы вправе выполнить все три указанные операции подряд, назвав этот одним большим вариантом использования «Приобрести товар», как показано на рис. 8-7. Описание варианта использования «Приобрести товар» может просто говорить о последовательном выполнении этих трех вариантов использования: «Искать по каталогу», «Добавить товар в корзину» и «Оплатить товары в корзине».

Чтобы это работало правильно, каждый вариант использования должен оставлять систему в состоянии, позволяющем пользователю немедленно начать следующий вариант использования. То есть выходные условия одного варианта использования должны удовлетворять предварительные условия

следующего варианта в последовательности. Аналогично в приложении обработки транзакций, таком как ПО банкомата, каждый вариант использования должен оставлять систему в состоянии, позволяющем выполнить следующую транзакцию.

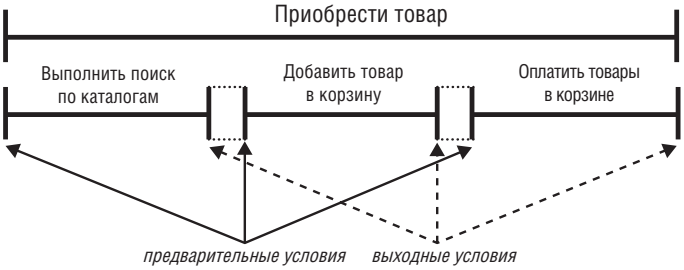


Рис. 8-7. Предварительные и выходные условия определяют границы отдельных вариантов использования, которые можно связать в цепочку для выполнения объемной задачи

Варианты использования и бизнес-правила

Варианты использования и бизнес-правила взаимосвязаны. Некоторые бизнес-правила ограничивают круг ролей, которые могут выполнять все или некоторые части варианта использования. Скорее всего, только пользователи с определенными уровнями привилегий могут выполнять определенные альтернативные направления. То есть правило может накладывать предварительные условия, которые система должна проверять, прежде чем разрешить пользователю продолжить работу. Бизнес-правила могут влиять на отдельные шаги нормального направления, задавая разрешенные входные значения или диктуя, какие вычисления должны выполняться. Представьте, что авиакомпания взимает с пассажиров дополнительную плату за места повышенной комфортности. Когда пассажир выполняет вариант использования по выбору нового места на веб-сайте авиакомпании, при выборе места повышенной комфортности соответствующее бизнес-правило увеличит цену билета. При определении варианта использования запишите идентификаторы всех известных бизнес-правил, которые влияют на этот вариант, а также укажите, на какую часть варианта использования влияют те или иные правила.

При изучении вариантов использования вы можете обнаружить бизнес-правила. Когда химики, участвующие в выявлении требований для системы Chemical Tracking System, обсуждали вариант использования, предусматривающий просмотр находящегося в системе заказа, один из них сказал: «Фред не должен видеть мои заказы, а я не должен видеть заказы Фреда». То есть они сформулировали бизнес-правило: пользователь может видеть только те заказы, которые создал сам. Иногда бизнес-правило обнаруживается во время процесса выявления и анализа требований, бывает, что уже существующие в организации правила обнаруживаются в процессе обсуждения, а может быть и так, что вы уже знаете о существующих правилах, которые должна соблюдать система.

Определение вариантов использования

Существует несколько способов определения вариантов использования (Ham, 1998; Larman, 1998):

- сначала определить действующих лиц, затем бизнес-процессы, поддерживаемые системой, и варианты использования для действий, в которых участвуют действующие лица и система;
- выразить бизнес-процессы в терминах определенных сценариев, обобщить сценарии в варианты использования и определить действующих лиц для каждого варианта;
- используя описание бизнес-процесса, задать вопрос: «Какие задачи должна система выполнить, чтобы реализовать этот процесс или преобразовать входные данные в выходные?» Эти задачи могут быть вариантами использования;
- определить внешние события, на которые система должна реагировать, а затем соотнести эти события с участвующими лицами и конкретными вариантами использования;
- применить CRUD-анализ (Create, Read, Update, Delete — создание, чтение, обновление, удаление) для определения сущностей данных, которым нужны варианты использования для создания, чтения, обновления, удаления или других операций с данными (см. главу 13);
- проанализировать контекстную диаграмму, задавшись вопросом: «Какие цели нужно каждой из внешних сущностей достичь с использованием системы?»

В команде разработки Chemical Tracking System применили первый подход — он описывается более детально далее в этой главе. Три бизнес-аналитика провели несколько семинаров по вариантам использования, которые и проводились примерно два раза в неделю по два часа. Для выявления требований они выбрали семинары отчасти из-за того, что никто из них ранее не применял подход на основе вариантов использования, поэтому им нужно было учиться вместе. Кроме того, они видели большую пользу от взаимодействия в группе в формате семинара, чем при использовании отдельных интервью. Члены различных классов пользователей участвовали в разных, параллельных семинарах, проводимых разными бизнес-аналитиками. Это хорошо работало, потому что лишь несколько вариантов использования были общими для нескольких классов пользователей. На каждом семинаре присутствовали сторонник продукта от класса пользователей, другие выбранные представители пользователей, а также разработчик. Участие в семинарах позволило разработчикам еще на ранней стадии получить представление о создаваемом продукте. Кроме того, они возвращали собравшихся к реальности, когда те предлагали невыполнимые требования.

До начала семинаров аналитики попросили пользователей продумать, какие задачи те собираются выполнять с помощью новой системы. Каждая

такая задача становилась потенциальным вариантом использования. Это подход «снизу вверх» к выявлению вариантов использования, который дополняет стратегию «сверху вниз» для определения всех бизнес-процессов, которые должна поддерживать система, и извлечения из них вариантов использования. Сравнение списков вариантов использования, полученных в этих разных процессах, снижает вероятность упустить какой-то вариант использования.

Несколько предложенных вариантов, как выяснилось, выходят за границы проекта, поэтому далее они не обсуждались. По мере изучения вариантов использования группе стало ясно, что некоторые варианты связаны между собой сценариями, которые можно объединить в один более общий вариант использования. Также группе удалось выявить дополнительные варианты использования, не входившие в первоначальный набор. Будьте готовы к выполнению подобных корректировок по ходу работы.

Некоторые участники предложили варианты использования, не сформулированные как задачи, например «Данные по безопасному хранению материалов». Название варианта использования должно указывать на решаемую задачу, поэтому в названии необходим глагол. Что клиент хочет: просмотреть, напечатать, заказать, отправить по электронной почте, исправить, удалить или создать данные по безопасному хранению материала? Иногда предложенный вариант использования — это всего лишь одно действие, которое выполняет действующее лицо как часть варианта использования, например «сканировать штрих-код». Аналитик должен выяснить, какую цель подразумевал пользователь, когда предлагал это. Он может задать такой вопрос: «Сканируя штрих-код на контейнере с химикатами, что вы пытаетесь сделать?» Предположим, в ответ он услышит: «Это необходимо для того, чтобы я мог зарегистрировать этот химикат за своей лабораторией». (Обратите, что это формулируется в стиле пользовательской истории.) Следовательно, настоящий вариант использования выглядит как-то так: «Зарегистрировать химикат в лаборатории». Сканирование штрих-кода — только один из этапов взаимодействия действующего лица и системы, регистрирующей химикат в лаборатории.

Не стоит углубляться в подробный анализ первого же предложенного кем-то варианта использования. Соберите о каждом варианте использования ровно столько информации, сколько необходимо для определения их приоритетов и выполнения начального назначения вариантов использования или их частей на будущие выпуски или итерации. После этого можно изучать самые приоритетные варианты использования, те что назначены на следующий цикл разработки, чтобы разработчики могли максимально быстро начать их реализацию. Менее приоритетные варианты использования могут подождать детализации до момента, непосредственно предшествующего их реализации по плану. Такую стратегию следует применять по отношению к пользовательским историям в проектах гибкой разработки.

Внимание! Не пытайтесь включить в вариант использования каждое появившееся требование. С помощью вариантов использования удается выявить большинство, но никак не все, функциональные требования. Если бизнес-аналитик уже знает об определенной функциональности, которую нужно реализовать, нет особого смысла в создании варианта использования просто для хранения этой функциональности.

Анализ вариантов использования

Участники семинара, посвященного сбору требований для системы Chemical Tracking System, начинали каждое обсуждение с определения действующего лица, которое получит пользу от данного варианта использования, и краткого описания этого варианта. Выяснив частоту использования, вы на ранних стадиях получите представление о требованиях по параллельному использованию и необходимых мощностях. Предварительные и выходные условия корректировались по мере появления новой информации в процессе дискуссии.

Далее аналитики спрашивали участников, как те себе представляют взаимодействие с системой в процессе выполнения задачи. Установленная последовательность действий лиц и реакции системы определялись как нормальное направление. Хотя у каждого участника было свое представление об интерфейсе, группа смогла выработать общее понимание важнейших этапов диалога системы и пользователя.

Придерживаясь границ

Изучая вариант использования, нормальное направление которого состояло из восьми этапов, я понял, что выходные условия удовлетворяются уже после этапа 5. Следовательно, этапы 6, 7 и 8 были излишними, так как выходили за границы варианта использования. Точно так же предварительные условия варианта использования должны быть удовлетворены до начала этапа 1. Изучая описание варианта использования, убедитесь, что предварительные и выходные условия ограничивают его надлежащим образом.

Бизнес-аналитик фиксировал действия конкретных лиц и реакцию системы на отдельных клейких листочках, которые затем прикреплял на схему. Клейкие листочки хорошо подходят для таких семинаров. В процессе обсуждения их легко перемещать, группировать и заменять. Возможен и другой способ проведения семинара — в ходе обсуждения спроецировать с помощью компьютера шаблон варианта использования на большой экран и отредактировать его, хотя иногда это замедляет обсуждение. Команда, занимающаяся сбором информации, разработала аналогичные диалоги для определения альтернативных направлений и исключений. Многие условия исключений удавалось выяснить, когда аналитик задавал примерно такие вопросы: «Что должно произойти, если в текущий момент БД недоступна?» или «Что, если этого химиката нет в продаже?». На семинаре также удобно обсудить ожидания пользователей, касающиеся качества продукта, например времени реак-

ции и доступности, ограничений дизайна пользовательского интерфейса и требований по безопасности.

После того как участники семинара описали каждый вариант использования, и никто не предлагал уже никаких дополнительных вариантов, исключений или специальных требований, приступали к следующему варианту использования. Участники не пытались рассмотреть все варианты использования за одну марафонскую дистанцию или точно определить все детали каждого обсуждавшегося варианта использования. Они запланировали изучение с самых приоритетных вариантов использования и их последующее итеративное уточнение вплоть до реализации.

На рис. 8-8 показана последовательность этапов разработки вариантов использования системы Chemical Tracking System. После семинара аналитик задокументировал все варианты использования, применяя шаблон, показанный на рис. 8-3. При этом он самостоятельно определял, насколько полным должен быть шаблон для каждого варианта использования.

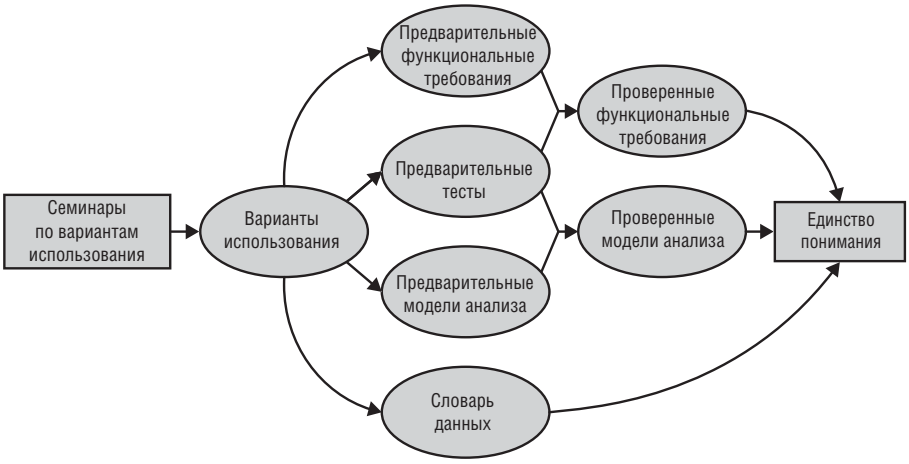


Рис. 8-8. Рабочие результаты процесса выявления вариантов использования

При написании этапов в направлениях вариантов использования избегайте формулировок, ссылающихся на конкретные взаимодействия в пользовательском интерфейсе. «Сотрудник указывает требуемый химикат» — вполне общая и не завязанная на пользовательский интерфейс формулировка. Она допускает много вариантов реализации намерения пользователя указать химикат, который нужно заказать: ввести идентификатор химиката, импортировать химическую структуру из файла, нарисовать структуру мышкой на экране (или стилусом на экране планшета) или выбрать химикат из списка. Слишком быстрый переход к конкретике взаимодействий ограничивает мысленные процессы участников семинара.

Часто варианты использования содержат дополнительную информацию или требования, которые не годятся ни для одного из разделов шаблона. Для таких данных предусмотрен раздел «Специальные требования»: сюда запи-

сывают соответствующие атрибуты качества, требования к производительности и др. Рано или поздно вся эта информация должна оказаться в спецификации требований или других документах требований. Также зафиксируйте любую информацию, которая может быть неочевидна пользователям, например необходимое для завершения варианта использования неявное взаимодействие одной системы с другой.

Проверка вариантов использования

На рис. 8-8 показано, что после каждого семинара бизнес-аналитики Chemical Tracking System выявляли функциональные требования к ПО на основе описаний вариантов использования (подробнее см. далее раздел «Варианты использования и функциональные требования»). Они также создали модели анализа, например диаграмму перехода состояний, показывающую все возможные состояния запроса химиката и все допустимые изменения состояний. Запрос химиката присутствует во многих вариантах использования, поэтому диаграмма объединяет информацию и операции, относящиеся ко многим вариантам использования. В главе 12 иллюстрируются несколько моделей анализа для системы CTS; диаграмма переходов состояний показана на рис. 12-3.

Спустя день или два после семинара, бизнес-аналитик раздал описания вариантов использования и функциональных требований участникам, чтобы те просмотрели их до начала следующего семинара. Эти неофициальные просмотры выявили множество ошибок: ранее не выявленные альтернативные направления, новые исключения, некорректные функциональные требования и пропущенные этапы диалога. В команде быстро поняли, что нужно делать перерыв между последовательными семинарами — хотя бы один день. Умственное расслабление, которое наступает, когда минует день или два после семинара, позволяет людям взглянуть на проделанную работу с новой точки зрения. Один аналитик, проводивший семинары каждый день, заметил, что участникам стало трудно находить ошибки в просматриваемых документах, потому что информация была еще слишком свежа в памяти. Они прокручивали в уме дискуссии, которые только что завершились, и не замечали ошибок.

> **Внимание!** Не ждите готовности спецификации требований, чтобы просить пользователей, разработчиков и других заинтересованных лиц заняться просмотром собранного материала. Каждая проверка позволяет улучшить последующие этапы работы над требованиями.

На ранних стадиях разработки Chemical Tracking System руководитель тестирования создал концептуальные варианты тестирования, не зависящие от специфики реализации и пользовательского интерфейса, на основе вариантов использования (Collard, 1999). Эти варианты тестирования помогли прийти команде к общему четкому пониманию того, как система должна функционировать при реализации определенных сценариев использования. Эти варианты тестирования позволили бизнес-аналитикам проверить, дей-

ствительно ли выявлены все функциональные требования, необходимые для выполнения пользователями каждого варианта использования. На заключительном семинаре, участники вместе провели критический анализ вариантов тестирования, чтобы убедиться, что одинаково понимают, как должны работать варианты использования.

Создание концептуальных вариантов тестирования на ранних этапах, как здесь, обходится значительно дешевле и выполняется быстрее, чем написание кода, создание части системы и выполнение тестирования для того, чтобы только после этого обнаружить проблемы с требованиями. Это похоже на гибкий подход к наполнению фактурой пользовательских историй с помощью приемочных тестов, но в команде Chemical Tracking System создали как функциональные требования, так и тесты. Их сравнение позволило обнаружить в них ошибки до написания какого-либо кода. Подробнее о создании тестов на основе требований см. главу 17.

В команде разработки Chemical Tracking System создали несколько представлений обнаруженных требований: список функциональных требований, набор соответствующих тестов и модели анализа — все они основаны на вариантах использования. Сравнение этих альтернативных представлений требований — мощный метод повышения качества (Wiegers, 2006). В команде тесты использовали для проверки функциональных требований, пытаясь обнаружить тесты, которые нельзя «выполнить» при имеющемся наборе требований, а также требования, которые не были охвачены тестами.

Если вы создадите лишь одно представление требований, приходится доверять ему. Вам не с чем его сравнить, чтобы выявить ошибки, пробели и различия в интерпретации. В проектах гибкой разработки обычно не документируют функциональные требования, предпочитая создавать приемочные тесты. Хотя обдумывание тестирования в процессе изучения требований и является отличной идеей в любом проекте, у вас все равно одно представление требований, которое следует считать правильным. Аналогично в командах традиционного подхода, где создают только один набор функциональных требований и оставляют тестирование на более поздние этапы проекта, есть только одно представление. Наилучшие результаты получаются при использовании разумной комбинации письменных требований, тестов, моделей анализа и прототипов.

Варианты использования и функциональные требования

Разработчики ПО не реализуют пользовательские или бизнес-требования. Они реализуют функциональные требования, определенные фрагменты поведения системы. Некоторые практики считают, что варианты использования это и есть функциональные требования. Однако мне случалось видеть, как возникали проблемы из-за того, что варианты использования просто передавались разработчикам для реализации. Варианты использования описывают точку зрения пользователя, его взгляд на внешнее, видимое поведение систе-

мы. В них не содержится всей информации, которая необходима разработчику для написания ПО. Так, человеку, использующему банковский автомат, не важно, какие неочевидные действия этот автомат должен выполнить, например взаимодействовать с компьютером банка. Подробности пользователю не видны, но разработчик должен знать о них. Даже у разработчиков, которым передали полные варианты использования, часто возникает много вопросов. Чтобы уменьшить эту неопределенность, я рекомендую бизнес-аналитикам ясно расписывать детали функциональных требований, необходимых для реализации каждого варианта использования (Arlow, 1998).

Многие функциональные требования не записаны в последовательности диалогов действующего лица и системы. Некоторые из них очевидны, например: «Система должна назначать уникальный порядковый номер каждому запросу». Нет смысла повторять эти детали в других местах, если они совершенно очевидно указаны в варианте использования. Другие функциональные требования не входят в описание варианта использования. В частности, типичная форма документирования вариантов использования не определяет, что должна делать система, когда предварительное условие *не выполняется*. Это пример того, что варианты использования часто не предоставляют всей необходимой информации, чтобы разработчик мог начать программирование. Отсутствующие требования выявляет бизнес-аналитик и доводит их до разработчиков и тестировщиков (Wiegers, 2006). Преобразование требований, как их видит пользователь, в форму, полезную разработчикам, — задача бизнес-аналитика, один из многих способов, позволяющих ему сделать проект действительно полезным.

В Chemical Tracking System варианты использования в основном применялись в качестве механизма для определения необходимых функциональных требований. Аналитики составляли только рабочие описания наименее сложных вариантов использования. Затем они выявляли все функциональные требования, которые после реализации позволяли действующим лицам выполнять все варианты использования, в том числе альтернативные направления и обработчики исключений. И далее документировали эти функциональные требования в спецификации требований к ПО, сформированной с учетом особенностей продукта.

Задокументировать функциональные требования, связанные с вариантом использованием, можно несколькими способами. Ни один из этих методов нельзя назвать идеальным, поэтому выберите тот, что лучше позволит вам документировать и управлять требованиями к ПО вашего проекта.

Только варианты использования

Вы можете включить функциональные требования непосредственно в описание каждого варианта использования, если выбор уже очевиден. Независимо от этого вам понадобится отдельная дополнительная спецификация для фиксирования нефункциональных требований и всей функциональности, которая не связана с конкретными вариантами использования. Кроме того,

несколько вариантов использования могут нуждаться в одном и том же функциональном требовании. Если для пяти вариантов использования потребуется аутентификация пользователя, не нужно писать пять разных блоков кода. Вместо их повторения, установите перекрестные ссылки на функциональные требования, присутствующие в нескольких вариантах использования. Варианты использования можно собрать в документе пользовательских требований.

Варианты использования и функциональные требования

Другая возможность — в довольно простой форме описать варианты использования и задокументировать функциональные требования, выявленные из каждого варианта использования в спецификации требований к ПО или хранилище требований. В этом случае вам придется установить соответствие между вариантами использования и связанными с ними функциональными требованиями. В такой ситуации при изменении варианта использования можно быстро найти соответствующие функциональные требования. Управлять такими связями лучше всего с помощью средства управления требованиями.

Только функциональные требования

Еще один вариант — упорядочить функциональные требования по вариантам использования или функциям и включить и то, и другое в спецификацию или хранилище требований. Именно его и применяли специалисты, работавшие над Chemical Tracking System, а также мы в нескольких проектах по разработке веб-сайтов. Мы описали большинство своих вариантов использования в очень сжатой форме, не заполняя весь шаблон, показанный на рис. 8-3. После этого подробности указывались с применением набора функциональных требований. При таком подходе отдельный документ пользовательских требований не создается.

Варианты использования и тесты

Если вы пишете подробные и спецификации вариантов использования, и функциональных требований, может обнаружиться дублирование, особенно в нормальном направлении. Нет смысла описывать одно и то же требование дважды. Поэтому есть еще один вариант — составить достаточно полные спецификации вариантов использования, а после этого написать приемочные тесты, проверяющие, правильно ли система обслуживает базовое поведение варианта использования, альтернативные успешные направления и другие вещи, которые могут пойти не так.

Каких подводных рифов следует опасаться при способе с применением вариантов использования

Как и любой другой способ разработки ПО, применение вариантов использования чревато возникновением множества проблем (Lilly, 2000; Kulak и Guiney, 2004). Следует опасаться следующих подводных рифов.

Слишком много вариантов использования Если вас захлестнул вал вариантов использования, вам, возможно, не удастся записать каждый из них с достаточным уровнем абстракции. Не создавайте отдельный вариант использования для каждого возможного сценария. Как правило, количество вариантов использования превышает количество бизнес-требований и функций, однако функциональных требований обычно бывает намного больше, чем вариантов использования.

Очень сложные варианты использования Однажды я изучал вариант использования объемом в четыре страницы, плотно заполненных описанием этапов взаимодействия, встроенной логики и условий ответвления. Документ казался весьма невразумительным. Мне встречались и более длинные варианты использования, занимающие много страниц. Вам не дано контролировать сложность бизнес-задач, но вы можете выбрать способ их представления в вариантах использования. Выберите один успешный способ выполнения варианта использования, с одной комбинацией значений правильных и ложных значений для различных логических решений и назовите его нормальным направлением. Другие логические ответвления определите как альтернативные направления и назначьте исключения для обработки неудачных ответвлений. Альтернативных направлений может быть множество, однако каждое должно быть кратким и понятным. Если направление состоит из 10-15 операций или больше, убедитесь, что оно действительно описывает один сценарий. Без особых на то оснований не разбивайте длинное, но корректное направление просто потому, что в нем очень много операций.

Включение дизайна в варианты использования Варианты использования должны описывать то, что пользователям необходимо выполнить с помощью системы, а не на то, как это будет выглядеть на экране. Сосредоточьтесь на концептуальном взаимодействии действующих лиц и системы. Например, правильная формулировка на этой стадии — «система предоставляет выбор», а не «система отображает раскрывающийся список». Не допускайте, чтобы дизайн пользовательского интерфейса диктовал направление разработки требований. Применяйте наброски экрана и карты диалоговых окон (см. главу 12), чтобы в визуальной форме представить взаимодействия исполнителя и системы, а не как подтверждение спецификации дизайна.

Включение определения данных в варианты использования Анализ вариантов использования естественным образом стимулирует обсуждение данных и того, какие элементы выступают как входные и выходные данные взаимодействий. Мне приходилось видеть варианты использования, которые включали в себя определения элементов данных и структур, которыми манипулируют в варианте использования. Это затрудняет для участников проекта поиск необходимой информации, поскольку неясно, в каком именно варианте использования оно содержится. В таких случаях возможен повтор определений, а значит, и их рассинхронизация, когда один экземпляр изменяется, а остальные нет. Соберите определения данных в словарь данных, созданный для всего проекта, как описано в главе 13.

Варианты использования, которые непонятны пользователям Если пользователи не видят связи описания вариантов использования со своими бизнес-процессами или задачами, это проблема. Составляйте варианты использования с точки зрения клиентов, а не системы и попросите клиентов проверить их. Делайте варианты использования максимально простыми, но тем не менее решающими задачу четкой и эффективной передачи информации.

Преимущества требований, основанных на вариантах использования

Сила подхода с применением вариантов использования и пользовательских историй в том, что он сосредоточен на поставленной задаче и пользователе. Пользователи будут более четко представлять, что же им даст новая система, если вы выберете способ, который фокусируется на функциях системы. При разработке нескольких интернет-проектов представители клиентов заявили, что варианты использования позволили им более четко представить, какие возможности должны получить посетители их веб-сайтов. А биз-нес-аналитики и разработчики смогли разобраться и в бизнесс-процессах пользователей, и в предметной области. Тщательное изучение этапов взаимодействия действующего лица и системы помогает еще на ранних стадиях разработки выявить неясности и неточности, а также позволяет составить тесты на основе вариантов использования.

Если вы будете заблаговременно определять требования и включать в них все мыслимые функции, то рискуете создать избыточные требования. Способ с применением вариантов использования позволяет выявить функциональные требования, которые позволят пользователям выполнять конкретные известные задачи. Так вы предотвратите появление «потерянных функций» — тех, что в ходе сбора информации казались весьма полезными, однако которыми никто не будет пользоваться из-за того, что они не связаны напрямую с решением задач пользователей.

Разработка пользовательских требований облегчает расстановку приоритетов требований. Высшим приоритетом обладают те функциональные требования, которые созданы на основе вариантов использования с высшим приоритетом. Высший приоритет вариантам использования или пользовательским историям назначается по следующим причинам:

- они описывают один из основных бизнес-процессов, активизируемых системой;
- многие пользователи часто обращаются к ним;
- их запросил привилегированный класс пользователей;
- они предоставляют возможности, необходимые для выполнения требований регулирующих органов;
- функции других систем зависят от их наличия.

> **Внимание!** Не тратьте много времени на обсуждение деталей вариантов использования, которые не будут реализованы в ближайшие месяцы или годы. Вероятнее всего, они изменятся еще до начала проектирования.

Существуют также и преимущества технического характера. С помощью варианта использования можно выявить некоторые важные объекты предметной области и их взаимоотношения. Разработчики, использующие объектно-ориентированные методы дизайна, могут преобразовать варианты использования в объектные модели, такие, как диаграммы классов и диаграммы последовательностей. По мере того как с течением времени изменяются бизнес-процессы, задачи, реализуемые посредством определенных вариантов использования, также изменяются. Если вы проследите функциональные требования, дизайн, код и тесты вплоть до их истоков — пожеланий клиента, вам будет легче внести эти изменения бизнес-процессов во всю систему.

Что дальше?

- Составьте несколько вариантов использования для вашего текущего проекта на основании шаблона на рис. 8-3. Включите в них все альтернативные направления развития и исключения. Определите функциональные требования, которые позволят пользователю успешно завершить каждый вариант использования. Проверьте, не включены ли уже все эти функциональные требования в спецификации требований к ПО.
- Если в вашей организации планируют внедрить приемы гибкой разработки, попробуйте написать один вариант использования как одну или набор пользовательских историй, чтобы оценить различие между двумя подходами.
- Пройдите вариант использования, пытаясь на каждом этапе вывести необходимые функциональные требования, а также предварительные и выходные условия, бизнес-правила и другие требования.
- Проверьте вариант использования с клиентами, чтобы убедиться в правильности определения этапов, а также в том, что рассмотрены все вариации основного направления, учтены и обработаны все исключения, которые клиенты посчитали разумными.

Глава 9

Игра по правилам

«Привет, Тим, это Джеки. У меня проблема с запросом химиката при помощи системы Chemical Tracking System. Мой менеджер посоветовал обратиться к тебе. Он сказал, что ты был сторонником продукта и занимался требованиями к этой системе».

«Да, это правда, — ответил Тим. — А в чем проблема?»

«Мне нужно еще немного фосгена для красок, которыми я сейчас занимаюсь,— сказала Джекки, — но система не принимает мой запрос и сообщает, что я уже больше года не посещала занятий по работе с опасными химикатами. О чем это она? Я годами работала с фосгеном — и никаких проблем. Почему же мне нельзя получить немного фосгена?»

«Я надеюсь, ты в курсе, что Contoso требует от сотрудников ежегодно прослушивать курс техники безопасности при работе с опасными химикатами, — заметил Тим. — Такова корпоративная политика, основанная на требованиях Закона о гигиене и безопасности труда на рабочем месте (OSHA); система контроля химикатов лишь проводит ее в жизнь. Я знаю, что раньше кладовщики давали тебе все, что требовалось, но теперь это невозможно. Извини за неудобства, но мы должны подчиняться правилам. Тебе придется пройти обучение, чтобы система выполнила твой запрос».

В любой организации действует широкий набор корпоративных политик, законов и промышленных стандартов. Такие отрасли, как банковское дело, авиация и производство медицинской техники, подчиняются требованиям государственных органов. Все эти контролирующие принципы в целом называются *бизнес-правилами* (business rules). Зачастую за соблюдением правил и процедур следят люди, но во многих случаях их выполнение и соблюдение осуществляется средствами программных систем.

Большинство бизнес-правил берут начало вне контекста какой-либо конкретной программной системы. Такие правила, как корпоративная политика, требующая ежегодно прослушивать курс по работе с опасными химикатами, действуют, даже если приобретение и распространение химикатов осуществляется исключительно вручную. Стандартные правила отчетности существовали задолго до изобретения компьютеров. Тем не менее, бизнес-правила — один из основных источников функциональных требований к ПО,

поскольку они диктуют свойства, которыми должна обладать система для выполнения правил. Как показано на рис. 1-1 в главе 1, бизнес-правила могут быть источником нескольких типов требований. Таблица 9-1 иллюстрирует и предоставляет примеры влияния бизнес-правил на несколько типов требований.

Табл. 9-1. Как бизнес-правила могут влиять на различные типы требований к ПО

Тип требования	Иллюстрация влияния бизнес-правила	Пример
Бизнес-требование	Требования государственных регулирующих органов могут создавать необходимые бизнес-цели в проекте	Система Chemical Tracking System должна поддерживать все федеральные и местные законы по обращению и утилизации химикатов, а также предоставлять в регулирующие органы все необходимые отчеты с периодичностью в пять месяцев
Пользовательское требование	Политики конфиденциальности диктуют, какие пользователи имеют право выполнять те или иные задачи в системе	Только менеджерам лабораторий разрешено генерировать отчеты о контактах с химикатами для любого пользователя, а не только для себя
Функциональное требование	Политика компании требует регистрации и одобрения всех поставщиков до оплаты первого счета	При получении счета-фактуры от незарегистрированного поставщика, система поставок должна отправить поставщику по электронной почте редактируемые PDF-версии формы регистрации поставщика и формы W-9
Атрибут качества	Требования государственных органов, таких как Управление по безопасности труда (OSHA) и Агентство по защите окружающей среды о хранении и использовании химикатов (EPA), могут диктовать требования по безопасности, которые должны быть реализованы в функциональности системы	Система должна хранить записи об обучении работе с опасными химикатами, чтобы право заказа опасных химикатов предоставлялось только пользователям, прошедших соответствующий курс

Люди иногда путают бизнес-*правила* с бизнес-*процессами* или бизнес-*требованиями*. Как вы видели в главе 5, бизнес-*требование* определяет желаемый результат или высокоуровневую цель организации, которая создает

или предоставляет программное решение. Бизнес-требования служат обоснованием для выполнения проекта. Бизнес-*процесс* описывает ряд действий по преобразованию входных данных в те выходные данные, которые нужны для получения определенного результата. Информационные системы часто автоматизируют бизнес-процессы, что может обеспечивать повышении эффективности и другие преимущества, которые обеспечивают выполнение сформулированных бизнес-требований. Бизнес-*правила* влияют на бизнес-процессы путем определения словаря, наложения ограничений, инициирования действий и управления порядком выполнения вычислений. Одно и то же бизнес-правило может применяться ко многим ручным и автоматизированным процессам, и по этой причине бизнес-правила лучше всего рассматривать, как отдельный массив информации.

Не все фирмы рассматривают собственные важнейшие бизнес-правила как ценность, которой они и являются на самом деле. В некоторых отделах могут задокументировать свои локальные правила, но во многих компаниях отсутствует унифицированный подход к документированию и хранению бизнес-правил в едином хранилище, доступном для ИТ-отдела. Отношение к жизненно важной информации как к корпоративным изустным знаниям создает массу проблем. Если эта информация не задокументирована и не хранится должным образом, она существует только в головах сотрудников. Бизнес-аналитик должен знать, к кому обратиться, чтобы узнать подробнее о правилах, которые влияют на проект. Сотрудники могут по-разному понимать правила, в результате чего разные программные системы могут по-разному выполнять одно и то же бизнес-правило или вообще игнорировать его. Наличие главного хранилища бизнес-правил упрощает доступ к этим правилам и согласованную реализацию приложений во всех проектах, на которые эти правила влияют.

> **Внимание!** Недокументированные бизнес-правила, известны только отдельным специалистам, а посему они теряются, когда последние увольняются или переходят на другую работу.

Например, в вашей организации скорее всего есть политики безопасности, управляющие доступом к информационным системам. Эти политики определяют минимальную и максимальную длину и разрешенные символы в пароле, диктуют частоту обязательного изменения пароля, число неудачных входов до блокировки учетной записи и т. п. Разрабатываемые в компании приложения должны соответствующим образом применять эти политики, то есть бизнес-правила. Возможность отследить место отдельных правил в коде облегчает обновление систем при изменении правил, например при определении другой частоты смены пароля. Это также способствует повторному использованию кода в проектах.

Классификация бизнес-правил

В группе Business Rules Group (2012) дали определение бизнес-правил с точки зрения как бизнеса, так и информационных систем:

- с точки зрения бизнеса: «бизнес-правило — это указание на существование обязательств относительно поведения, действия, принятого порядка или процедуры в определенной деятельности или отрасли»; (Должна быть явная мотивировка правила, а также методы обеспечении его выполнения, а также понимание последствий в случае нарушения правила.)
- с точки зрения информационной системы: «бизнес-правило — это указание, определяющее или ограничивающее определенный аспект бизнеса. Оно предназначается для установления бизнес-структуры или для управления и влияния на бизнес-деятельность».

Целые методологии разработаны специально для создания и документирования бизнес-правил и их применения в автоматизированных системах бизнес-правил (Ross, 1997; von Halle, 2002). Если вы не создаете систему, которая в значительной степени управляется бизнес-правилами, тщательно разработанная методология вам не нужна. Достаточно выявить и задокументировать относящиеся к вашей системе правила и связать их с конкретными требованиями, которые их реализуют.

Для организации бизнес-правил предлагается множество разных схем классификации (Ross, 2001; Morgan, 2002; von Halle, 2002; von Halle и Goldberg, 2010). Простейшая из них (рис. 9-1), из пяти типов бизнес-правил, годится в большинстве случаев. Шестая категория — термины: важные для бизнеса слова, фразы и аббревиатуры. Термины можно группировать по фактическим бизнес-правилам. Их также удобно хранить в словаре.

Рис. 9-1. Простая таксономия бизнес-правил

Вести согласованный свод бизнес-правил гораздо важнее при разработке продукта, чем горячо дискутировать о том, как их классифицировать. Вместе с тем таксономия помогает выявить бизнес-правила, о которых вы могли не знать. Классификация правил также дает представление о том, как их можно применить в приложении. Например, ограничения часто ведут к созданию функциональности применения ограничений, а активаторы операций обусловливают функциональность, предусматривающую выполнение определенных действий при определенных условиях. Давайте рассмотрим различные типы бизнес-правил, с которыми вам придется столкнуться.

Факты

Факты — это всего лишь верные утверждения о бизнесе на определенный момент времени. Они описывают связи и отношения между важными бизнес-терминами. Факты о сущностях данных, важных для системы, иногда применяют в моделях данных (подробнее о моделировании данных см. главу 13). Вот примеры фактов:

- на каждый химический контейнер нанесен уникальный штрих-код;
- оплачивается доставка каждого заказа;
- со стоимости доставки налог с продаж не берется;
- если лицо, приобретающее химикат по невозмещаемому авиабилету, изменяет маршрут доставки, ему потребуется уплатить дополнительный сбор;
- книги, размер которых превышает 16 дюймов, размещаются в разделе широкоформатных книг библиотеки.

Естественно, что в компании есть бесчисленное количество фактов. Если собирать не относящиеся к делу факты, бизнес-анализ быстро застопорится. Даже если эти факты правильные, не всегда очевидно, как команда разработчиков должна использовать эту информацию. Сосредоточьтесь на фактах, находящихся в границах проекта, и не старайтесь собрать весь массив бизнес-знаний. Старайтесь связать каждый факт с входными и выходными данными контекстной диаграммы, событиями системы, известными объектами данных или конкретными пользовательскими требованиями.

Ограничения

Ограничение определяет, какие операции не может выполнять система и ее пользователи. Вот некоторые слова и фразы, которые часто применяются при описании ограничивающего бизнес-правила: *должен*, *не должен*, *не может* и *только* определенные люди или роли могут выполнять определенные действия. Далее приводится несколько примеров ограничений разного происхождения.

Политики организации

- Договор о займе человека младше 18 лет должен подписывать один из его родителей или законный опекун, выступающий одновременно поручителем.
- Постоянный посетитель библиотеки может отложить для себя до 10 книг.
- В переписке по страховке не может отображаться больше 4 цифр номера социального страхования гражданина.

Государственные нормативы

- Все программы должны соответствовать государственным предписаниям, касающимся использования их людьми с ослабленным зрением.
- Экипажи коммерческих авиарейсов должны каждые 24 часа отдыхать не менее 8 часов.
- Если продление не предоставлялось, индивидуальные запросы на возврат федерального налога должны быть отправлены (по почтовому штемпелю) до полночи первого рабочего дня 14 апреля.

Отраслевые стандарты

- Лица, обращающиеся за ипотечной ссудой, должны отвечать требованиям Федерального управления жилищного строительства.
- Веб-приложения не должны содержать никаких HTML-тегов или атрибутов, не соответствующих стандарту HTML 5.

Так много ограничений

У проектов по разработке ПО множество разных ограничений. Менеджеры проекта должны соблюдать график, а также лимиты по персоналу и бюджету. Эти ограничения уровня проекта фиксируются в плане управления проектом по разработке ПО. Ограничения уровня продукта на проектирование и реализацию представляют ограничения, которые можно было бы ожидать оставленными на усмотрение людей, ответственных за построение решения. Такие ограничения, сужающие круг доступных разработчику возможностей, записаны в спецификации требований к ПО или спецификации архитектуры. Некоторые бизнес-правила налагают ограничения на деловые операции — они должны храниться в хранилище бизнес-правил. Когда эти ограничения отражаются в функциональных требованиях к ПО, следует указать соответствующие правила в качестве обоснования всех производных требований.

Ограничивающие бизнес-правила могут создавать определенные последствия для разработки ПО, даже если они непосредственно не выражаются в определенной функциональности. Представьте себе политику розничной точки продажи, разрешающую принимать обратно товары стоимостью свыше 50 долларов только контролерам и менеджерам. Если вы разрабатываете приложение точки продаж, которая будет использоваться сотрудниками магазина, это означает, что у каждого пользователя должен быть свой уровень привилегий. Приложение должно проверять, достаточно ли у текущего пользователя привилегий для выполнения определенных действий, таких как открытие выдвижного ящика с наличностью в кассовом аппарате для возврата денег покупателю за возвращенный товар.

Многие ограничительные бизнес-правила определяют, какие типы пользователей имеют доступ к определенным функциям, и лаконичным спосо-

бом документирования таких правил является матрица ролей и разрешений (Beatty и Chen, 2012). На рис. 9-2 показана такая матрица для различных пользователей информационной системы публичной библиотеки. Роли разделены на сотрудников библиотеки и прочих лиц. Функции системы разбиты на системные операции, операции с записями читателей библиотеки и операции с единицами хранения. Крестик в ячейке означает, что у указанной в столбце роли есть разрешение на выполнение операции, определенной в строке.

Активаторы операций

Правило, при определенных условиях инициирующее выполнение определенных действий, называется *активатором операции* (action enabler). Человек может выполнять эти действия вручную. Как вариант, правило может управлять некоторыми программными функциями, благодаря которым приложение при выполнении определенных условий реализует нужную модель поведения. Условия, определяющие выполнение операции, иногда представляют собой сложную комбинацию значений «истина» и «ложь», выполняющихся для нескольких отдельных условий. Таблица принятия решений (как в главе 12) — это выразительный способ документирования активирующих операции бизнес-правил расширенной логики. Выражение вида «Если *<некоторое условие верно или наступило определенное событие>*, то *<что-то произойдет>*» — это ключ, который описывает активатор операции. Вот несколько примеров таких бизнес-правил, активирующих определенные операции, в системе Chemical Tracking System:

- Если в наличии имеются контейнеры с запрошенным химикатом, их следует предложить запросившему химикат лицу.
- В последний день квартала должны генерироваться отчеты для Управления по безопасности труда (OSHA) и Агентства по защите окружающей среды о хранении и использовании химикатов (EPA) в этом квартале.
- Если срок хранения контейнера с химикатом истек, об этом необходимо уведомить лицо, у которого в данный момент находится контейнер.

В компаниях часто разрабатывают политики, призванные повысить доходность. Подумайте, как книжный интернет-магазин может применить следующие бизнес-правила для стимулирования импульсивных покупок после того, как посетитель уже решил приобрести определенный товар:

- Если клиент заказал книгу автора, написавшего несколько книг, перед оформлением заказа нужно клиенту предложить приобрести другие книги того же автора.
- После добавления клиентом книги в корзину отобразить аналогичные книги, которые приобрели другие клиенты вместе с этой книгой.

Матрица ролей и разрешений	Сотрудник библиотеки	Администратор	Библиотекарь	Библиотечный консультант	Не сотрудник библиотеки	Волонтер	Читатель
Системные операции							
Вход в библиотечную систему		X	X	X			
Создание штатных сотрудников		X					
Печать списка забронированнных книг		X	X	X			
Карточки читателей библиотеки							
Просмотр карточки читателя		X	X				
Редактирование карточки читателя		X	X				
Просмотр собственной карточки читателя		X	X	X		X	X
Выдача библиотечной карточки		X	X				
Прием штрафа		X	X				
Операции с единицами хранения							
Поиск по библиотечному каталогу		X	X	X		X	X
Резервирование единицы хранения		X	X				
Снятие единицы хранения с резерва		X	X	X		X	
Пересылка единицы хранения в другой филиал		X	X	X		X	
Бронирование единицы хранения		X	X	X		X	X

Рис. 9-2. Ограничивающие бизнес-правила иногда могут представляться в виде матрицы ролей и разрешений

Как мне отказали

Недавно я хотел заказать, используя часть моих «миль постоянного клиента» авиакомпании Blue Yonder Airlines, билет для моей жены Крис. Когда я попытался сделать это, сервер BlueYonder.com сообщил мне, что из-за возникшей ошибки выдача билета невозможна, и предложил немедленно позвонить в авиакомпанию. Менеджер по бронированию, который поднял трубку, сообщил мне, что авиакомпания не может оформить премиальный билет за налетанные мной мили по обычной или электронной почте, поскольку у меня и Крис разные фамилии. Для получения билета мне пришлось ехать в кассу аэропорта и предъявлять удостоверение личности.

Данный инцидент вызван следующим бизнес-правилом, которое звучит примерно так: «Если фамилия пассажира отличается от фамилии лица, на которого оформляется премиальный билет за налетанные мили, лицо, оформляющее билет, должно получить его лично». По-видимому, основание для этого бизнес-правила — предотвращение мошенничества. ПО веб-сайта Blue Yonder исправно применяет данное правило, но создает неудобства клиентам. Вместо того чтобы сообщить мне о правиле, касающемся разных фамилий и действиях, которые мне следует выполнить, система просто выводит тревожное предупреждение об ошибке. В результате я и менеджер по бронированию потратили время на ненужный теле-

фонный разговор. Плохо продуманные реализации бизнес-правил могут вызвать неприятие клиентов и, следовательно, отрицательно скажутся на вашем бизнесе.

Выводы

Вывод (inference), иногда его еще называют *предположительным знанием* или *производным фактом*, создает новый факт на основе других фактов. Выводы зачастую записывают в формате «если-то», применяемом также при записи бизнес-правил, активирующих операции; тем не менее, раздел «то» вывода заключает в себе факт, а не действие. Вот несколько примеров выводов:

- Если платеж не поступил в течение 30 календарных дней с момента отправки счета, то счет считается просроченным.
- Если поставщик не может поставить заказанный товар в течение пяти дней с момента получения заказа, то заказ считается невыполненным.
- Химикаты с токсичностью агента LD50 ниже 5 мг на килограмм массы мыши считаются опасными.

Вычисления

Пятый класс бизнес-правил определяют *вычисления* (computations), преобразующие существующие данные в новую информацию с использованием математических формул и алгоритмов. Многие вычисления выполняются по внешним для предприятия правилам, например по формулам удержания подоходного налога. Ниже дано несколько примеров вычислительных бизнес-правил, выраженных в текстовой форме.

- плата за доставку по суше в пределах страны для заказов весом свыше 2 фунтов составляет 4,75 доллара плюс 12 центов за полную или неполную унцию;
- общая стоимость заказа вычисляется как сумма стоимости всех заказанных товаров, за вычетом скидок на количество, плюс государственные и местные налоги, действующие в округе, куда будет доставлен товар, плюс стоимость доставки и плюс необязательный страховой сбор;
- цена единицы товара снижается на 10% при заказе от 6 до 10 единиц, на 20% — при заказе от 11 до 20 единиц и на 30% — при заказе свыше 20 единиц.

Представление деталей вычислений на обычном языке может быть многословным и запутанным. Как вариант, их можно представить в символьной форме, например в виде математического выражения или таблицы правил, — такое представление прозрачнее и проще в сопровождении. Таблица 9-2 представляет правило предоставления скидок в более понятной форме.

Табл. 9-2. Использование таблицы для вычислений, необходимых для представления бизнес-правил

Идентификатор	Количество приобретаемых единиц товара	Скидка, %
DISC-1	1—5	0
DISC-2	6—10	10
DISC-3	11—20	20
DISC-4	свыше 20	30

Внимание! При написании бизнес-правил требований следите, чтобы границы диапазонов не перекрывались. Очень легко по невнимательности определить диапазоны как 1–5, 5–10 и 10–20, что привносит неоднозначность, потому что непонятно, в какой диапазон попадают значения 5 и 10.

Атомарные бизнес-правила

Представьте себе, что вы обратились к библиотекарю с вопросом: «На сколько времени я могу взять DVD-диск?» Библиотекарь отвечает: «Диск DVD или Blu-Ray можно взять на неделю, после чего можно продлить два раза по три дня, но только если другой читатель не забронировал диск». Ответ библиотекаря основан на бизнес-правилах библиотеки. Но в одной ее фразе присутствуют несколько правил. Такие сложные бизнес-правила могут быть сложными для понимания и поддержки. Также сложно проверить выполнение всех возможных условий. Если несколько частей функциональности относятся к сложному правилу, при изменении правила в будущем поиск и изменение соответствующего кода может занять много времени.

Более правильная стратегия заключается в написании бизнес-правил на атомарном уровне, не объединяя их в одно сложное правило. Это позволяет делать правила короткими и простыми. Это также упрощает повторное использование правил, изменение и объединение их в различных сочетаниях. Для записи предположительного знания и активирующих операции бизнес-правил в атомарном виде не используйте логику «или» в левой части конструкции «если-то» и избегайте логику «и» — в правой (von Halle, 2002). Указанное сложное библиотечное правило можно разбить на несколько атомарных бизнес-правил, которые показаны в Таблице 9-3. (Примененная в таблице 9-3 иерархическая система обозначений описана в главе 10.) Эти бизнес-правила называются *атомарными* (atomic), потому что их нельзя разбить на более мелкие правила. Скорее всего у вас получится много атомарных бизнес-правил, и ваши функциональные требования будут основываться на разных их сочетаниях.

Просто для иллюстрации того, как атомарные бизнес-правила упрощают обслуживание, скажем, что при появлении видео-технологии следующего поколения или при удалении всех DVD-дисков из библиотеки достаточно

будет всего лишь обновить правило Video.Media.Types, а остальные правила менять не придется.

Табл. 9-3. Некоторые атомарные бизнес-правила библиотеки

Идентификатор	Правило
Video.Media.Types	Диски DVD и Blu-ray относятся к видеоматериалам
Video.Checkout.Duration	Видеоматериалы можно арендовать в библиотеке на неделю
Renewal.Video.Times	Аренду видеоматериалов можно продлять до двух раз
Renewal.Video.Duration	При продлении аренды видеоматериала срок сдачи продляется на три дня
Renewal.HeldItem	Читатель не может продлить аренду видеоматериала, зарезервированного другим читателем

Документирование бизнес-правил

Поскольку бизнес-правила зачастую влияют на множество приложений, организациями следует управлять этими правилами на корпоративном уровне, а не на уровне проектов. Для начала достаточно простого каталога бизнес-правил. Если вы используете средство управления требованиями, можете хранить бизнес-правила в виде требований при условии, что они доступны из всех ваших проектов разработки ПО. Большим организациям, а также компаниям, деловые операции и информационные системы которых в значительной степени регулируются и управляются бизнес-правилами, следует создать базу данных таких правил. Некоторые системы управления бизнес-правилами содержат механизмы, позволяющие автоматизировать реализацию правил в приложениях. Группа Business Rules Group (2012) публикует список продуктов для управления бизнес-правилами. По мере того, как вы при работе над приложением определяете новые правила, добавляйте их в каталог, а не вписывайте в документацию конкретного приложения или, что еще хуже, только в его код. Серьезную опасность представляет неправильное управление и соблюдение правил, относящихся к безопасности, защите, финансам и выполнению требований законодательства и регулирующих органов.

> **Внимание!** Не надо делать свой каталог бизнес-правил более сложным, чем необходимо. Используйте простейшую форму документирования правил; это гарантирует, что команда разработчиков сможет эффективно использовать их. За хранилище правил должны отвечать бизнес-подразделения, а не ИТ-отдел или команда проекта.

По мере приобретения опыта выявления и документирования бизнес-правил стоит подумать о применении структурированных шаблонов для определения правил разных типов (Ross, 1997; von Halle, 2002). В этих шаблонах описываются образцы ключевых слов и разделов, позволяющих единообразно структурировать правила. Кроме того, они упрощают хранение правил в БД, серийных инструментах для управления бизнес-правилами

или в подсистеме управления правилами. Наборы взаимосвязанных правил можно также представлять с помощью таких средств, как дерева или таблицы принятия решений (особенно, в случае сложной логики) и матрицы разрешений. Для начала попробуйте использовать простой формат документирования правил, показанный в табл. 9-4 (Kulak и Guiney, 2004).

Табл. 9-4. Пример каталога бизнес-правил

Идентификатор	Определение правила	Тип правила	Статичное или динамическое	Источник
ORDER-5	Если клиент заказал книгу автора, написавшего несколько книг, перед оформлением заказа нужно предложить приобрести другие книги того же автора	Активатор операции	Статичное	Маркетинговая политика XX
ACCESS-8	Во всех изображениях веб-сайта должен быть альтернативный текст для использования устройствами электронного чтения, применяемыми слабовидящими пользователями	Ограничение	Статичное	Стандарты ADA для дизайна с расчетом на ограниченные возможности
DISCOUNT-13	Размер скидки вычисляется на основе размера заказа согласно данным табл. BR-060.	Вычисление	Динамическое	Корпоративная ценовая политика XXX

Как видно из этой таблицы, присвоив каждому правилу идентификатор, вы сможете отслеживать, от какого правила происходит то или иное функциональное требование. Например, некоторые шаблоны вариантов использования содержать поле для указания бизнес-правила, которое влияет на этот вариант использования. Вместо включения определения правила в варианте использования можно просто вводить идентификаторы соответствующих правил. Каждый такой идентификатор служит указателем на основной экземпляр бизнес-правила. В этом случае не нужно беспокоиться о том, что спецификация варианта использования может устареть из-за изменения правила.

Столбец «Тип правила» указывает, чем является правило: фактом, ограничением, активатором операции, выводом или вычислением. Поле «Статичное или динамическое» указывает, насколько вероятно изменение правила с течением времени. Эта информация нужна разработчикам. Если они будут знать, что определенные правила могут меняться, то смогут так структурировать ПО, чтобы изменившуюся функциональность или данные можно было легко обновить. Порядок исчисления налога на прибыль меняется по крайней мере раз в год. Если разработчик разместит информацию о расчете налога в таблице или базе данных, а не жестко пропишет ее в коде, будет намного проще изменить значения при необходимости. Можно спокойно документировать законы природы, такие как вычисления, основанные на законах термодинамики, а «человеческие» законы намного более переменчивы.

Законы эшелонирования

Системы управления воздушным движением (АТС) гарантируют минимальное эшелонирование (то есть интервалы) между самолетами по четырем видам направлений: вертикальное, продольное, боковое и временное — это нужно для предотвращения столкновений. Бортовая авионика, пилоты, диспетчеры на земле и сама система АТС должны собирать информацию о траектории движения и скорости из сотен источников, чтобы вовремя спрогнозировать, когда два самолета могут сблизиться на опасное расстояние. Многие бизнес-правила определяют минимальные интервалы эшелонирования. Эти правила динамичны: они периодически меняются вместе с изменениями в технологиях (например, вместе с заменой радаров системой GPS), а также меняются нормативные документы. Это подразумевает, что система должна уметь регулярно адаптироваться к новым правилам, проверять согласованность и полноту правил, а также переключаться на новые правила одновременно с пилотами и диспетчерами. В одном проекте системы АТС текущий набор правил жестко прописали в коде, считая, что они статичны. Потребовалась серьезная переделка, когда заинтересованные лица осознали необходимость регулярных изменений в этих жизненно важных для безопасности правилах.

Последний столбец в таблице 9-4 указывает на источник правила. Такими источниками могут быть корпоративные или управленческие политики, специалисты предметной области и другие лица, а также документы, такие как нормативные документы и законы. Знание источника помогает понять, куда обращаться, если нужна дополнительная информация о правиле и нужно подробнее узнать об изменениях.

Выявление бизнес-правил

Просто задавая пользователем вопрос: «Каковы ваши бизнес-правила?», практически невозможно получить нужную информацию, точно так же, как

не удается собрать нужные сведения о требованиях, задавая вопрос: «Чего вы хотите?» Иногда бизнес-правила создаются по ходу работы, в некоторых случаях — в процессе обсуждения требований, а бывает так, что за ними надо специально охотиться. Полный процесс выявления бизнес-правил описан Барбарой фон Хале (Barbara von Halle, 2002). Вот несколько стандартных мест и способов поиска правил (Boyer и Mili, 2011):

- «Общеизвестные знания» организации — их обычно можно собрать у людей, которые давно работают в компании и знают все подробности ее работы.
- Унаследованные системы, в требованиях и коде которых реализованы бизнес-правила. Для этого проводят обратный инжиниринг оснований, на которых базируются требования или код, чтобы понять связанные с ними правила. В результате иногда удается получить полные знания о бизнес-правилах.
- Моделирование бизнес-процессов, что требует от аналитика искать правила, которые влияют на каждый шаг процесса: ограничения, инициирование событий, правила вычислений и связанные факты.
- Анализ имеющейся документации, в том числе спецификаций требований из предыдущих проектов, нормативные документы, отраслевые стандарты, документы корпоративных политик, контракты и бизнес-планы.
- Анализ данных, таких как разные состояния, которые могут быть у объектов данных, и условия, в которых пользователь или системное событие может изменить состояние объекта. Эти последовательности могут также представляться в виде матрицы ролей и разрешений, похожей на показанную на рис. 9-2, для предоставления информации о правилах уровней привилегий пользователей и безопасности.
- Отделы нормативно-правового соответствия в компаниях, в которых создаются системы и которые подчиняются определенным правилам и нормативам.

Одного факта того, что вы нашли какие-то бизнес-правила в этих источниках, недостаточно, чтобы предполагать, что они применимы к вашему текущему проекту и что они вообще все еще имеют смысл. Формулы вычислений, реализованные в коде унаследованных приложений, могут быть устаревшими. Не забудьте проверить, не нуждаются ли в обновлении правила, полученные из старых документов и приложений. Оцените область применения обнаруженных правил. Являются ли они локальными по отношению к проекту или охватывают определенную область предприятия или всю организацию?

Часто заинтересованные лица проекта уже знают о бизнес-правилах, которые оказывают влияние на приложение. Некоторые сотрудники имеют дело с вполне определенными типам или классами правил. Если в вашей ситуации дела обстоят именно так, выясните, кто эти люди, и привлеките их к обсуждению. Бизнес-аналитик может уточнять бизнес-правила во время

выявления требований, которые также определяют другие артефакты и модели требований. На интервью и семинарах бизнес-аналитик может поинтересоваться обоснованием требований и ограничений, выдвигаемых пользователями. Нередко выясняется, что они определяются бизнес-правилами. На рис. 9-3 показано несколько возможных источников правил. Кроме того, здесь перечислены некоторые вопросы, которые бизнес-аналитик может задавать участникам при обсуждении различных проблем.

Рис. 9-3. Выявление бизнес-правил с помощью вопросов с разных точек зрения

Бизнес-правила и требования

Идентифицировав и задокументировав бизнес-правила, определите, какие из них необходимо реализовать в программном продукте. Иногда бизнес-правила и соответствующие им функциональные требования весьма похожи. Тем не менее правила — это внешние положения политики, которые требуется реализовать в ПО; следовательно, они управляют функциональностью системы. Каждый бизнес-аналитик должен решить, какие из имеющихся правил относятся к его приложению, какие необходимо реализовать в нем и как именно.

Вспомним ограничивающее правило для системы Chemical Tracking System, согласно которому запрос пользователя о выделении опасных химикатов удовлетворяется, только если пользователь прослушал курс по работе с ними за соответствующий период. Аналитику придется определить соответствующие этому правилу функциональные требования в зависимости от того, доступна ли БД записей об обучении в интерактивном режиме или нет.

Если да, система может просто просмотреть запись об обучении пользователя и решить, принять или отклонить его запрос. Если записи в интерактивном режиме недоступны, система может временно сохранить запрос на химикат и по электронной почте обратиться к руководителю курса, который, в свою очередь, подтвердит или не подтвердит запрос. Правило одинаково для обеих ситуаций, однако функциональные требования к ПО — действия, предпринимаемые системой, если бизнес-правило встречается в ходе работы — различаются в зависимости от рабочей среды системы.

В качестве еще одной иллюстрации взгляните на следующие правила:

- **Правило №1 (активатор операции).** «Если срок хранения контейнера с химикатом истек, об этом необходимо уведомить лицо, у которого в данный момент находится контейнер».
- **Правило №2 (вывод).** «Считается, что срок хранения контейнеров с химикатами, разлагающимися на взрывоопасные составляющие, истекает через один год с даты изготовления».

Правило №1 служит источником функции системы, которая называется «Известить владельца химиката об истечении срока хранения». Дополнительное правило, типа правила №2, помогает системе определить, у каких контейнеров есть срок годности, чтобы можно было своевременно уведомить их владельцев. Например, открытый сосуд с эфиром становится небезопасным, потому что в присутствии кислорода образует взрывоопасную смесь. На основе такого правила становится ясно, что Chemical Tracking System должна отслеживать состояние контейнеров с химикатами, у которых есть срок годности, и информировать соответствующих людей о возврате контейнеров для безопасной утилизации. Бизнес-аналитик может вывести некоторые функциональные требований для этой функциональности, например такие:

- **Expired.Notify.Before** Если состояние контейнера с химикатом, у которого есть срок годности, не «Утилизирован», система должна уведомить текущего владельца контейнера за неделю до истечения срока годности контейнера.
- **Expired.Notify.Date** Если состояние контейнера с химикатом, у которого есть срок годности, не «Утилизирован», система должна уведомить текущего владельца контейнера в день истечения срока годности контейнера.
- **Expired.Notify.After** Если состояние контейнера с химикатом, у которого есть срок годности, не «Утилизирован», система должна уведомить текущего владельца контейнера неделю спустя после истечения срока годности контейнера.
- **Expired.Notify.Manager** Если состояние контейнера с химикатом, у которого есть срок годности, не «Утилизирован», система должна уведомить менеджера текущего владельца контейнера две недели спустя после истечения срока годности контейнера.

При обнаружении набора таких сильно похожих требований надо попытаться изложить их в виде таблицы, а не списка (Wiegers, 2006). Представление получается компактнее и проще для проверки, понимания и модификации. Оно также позволяет назначать требованиям более лаконичные метки, потому что таблица представляет только суффиксы, которые добавляются к меткам родительских требований. Вот альтернативное представление представленных выше четырех функциональных требований:

- **Expired.Notify.Date** Если состояние контейнера с химикатом, у которого есть срок годности, не «Утилизирован», система должна уведомить указанных в таблице людей в указанное время.

Идентификатор требования	Кого уведомлять	Когда уведомлять
.Before	Текущего владельца контейнера	За неделю до истечения срока годности
.Date	Текущего владельца контейнера	В день истечения срока годности
.After	Текущего владельца контейнера	Неделю спустя после истечения срока годности
.Manager	Менеджера текущего владельца контейнера	Две недели спустя после истечения срока годности

Сводим все воедино

Во избежание избыточности не дублируйте правила из каталога бизнес-правил в спецификации требований к ПО. Вместо этого спецификация требований к ПО должна ссылаться непосредственно на конкретные правила как на источник определенной функциональности или алгоритмов. Можно определять ссылки между функциональным требованием и соответствующими родительскими бизнес-правилами — всего существует три возможности.

- Если вы используете инструмент управления требованиями, создайте атрибут требования «Источник» и укажите правила, из которых выведены функциональные требования (см. главу 27).
- Определите связи между функциональными требованиями и соответствующими бизнес-правилами в матрице связей требований или матрице сопоставлений требований (Beatty и Chen, 2012). Проще всего это делать, если бизнес-правила хранятся в том же хранилище, что и требования (см. главу 29).
- Если бизнес-правила и требования хранятся в текстовых файлах или электронных таблицах, создайте гиперссылки от идентификаторов бизнес-правил в требованиях на описания бизнес-правил, хранящиеся в других местах. Не забывайте, что гиперссылки могут «ломаться», если местоположение массива правил меняется.

Эти ссылки обеспечивают актуальность требований при изменении, а требования просто ссылаются на основную копию правила. При изменении правила можно по связанному идентификатору найти требования — или реализованную функциональность, — которые надо изменить. Такое использование ссылок упрощает повторное использование одного правила в нескольких местах и проектах, потому что правила не похоронены в документации отдельного приложения. Однако для детального просмотра правил разработчику, читающему спецификацию требований к ПО, придется следовать по перекрестным ссылкам, чтобы получить доступ к деталям правила. Это компромисс, на который приходится идти, если вы не собираетесь дублировать информацию (Wiegers, 2006).

Как и в случае с другими особенностями создания требований, это не так просто создать отличное решение для управления бизнес-правилами, которое работало бы во всех ситуациях. Когда вы начнете активно выявлять, фиксировать и применять бизнес-правила, всем заинтересованным лицам станет понятнее обоснование тех или иных выбранных вами вариантов разработки приложения.

Что дальше?

- Попробуйте найти в своем проекте по крайней мере по одному бизнес-правилу каждого типа, указанного в таксономии на рис. 9-1.
- Начните заполнять каталог бизнес-правилами, которые имеют отношение к вашему текущему проекту. Классифицируйте правила в соответствии со схемой на рис. 9-1 и обращая особое внимание на источник каждого из них.
- Создайте матрицу связей, указывающую, какие функциональные требования реализуют каждое из выявленных вами бизнес-правил.
- Выясните обоснование каждого из ваших функциональных требований, чтобы выявить другие, неявные бизнес-правила.

Глава 10

Документирование требований

В начале работы над крупным проектом по созданию следующего поколения флагманского продукта в компании по разработке ПО старший менеджер собрал примерно 60 сотрудников на продолжающееся целый день выездное мероприятие под девизом «семинар по изучению потребностей клиента». Эти сотрудники работали с организаторами над выработкой идеи нового продукта. Менеджер собрал результаты этих мозговых штурмов в 100-страничный документ. Он назвал это спецификацией требований, но на самом деле это было ни чем иным как просто кучей информации.

Информация, полученная в процессе этого семинара, не была разбита по категориям, логически структурирована, проанализирована или как-то иначе обработана в то, что можно было бы назвать предлагаемым программным решением. Разработчики были не в состоянии разобраться в этом огромном массиве идей и понять, что им нужно знать о новом продукте. Конечно, в этой куче были зерна ценных требований, но они были похоронены под огромным количеством шелухи. Простой сбор сырых идей и потребностей в длинный список — далеко не самый эффективный способ документирования и передачи требований к ПО.

Четкая и эффективная коммуникация — основополагающий принцип разработки требований — коммуникации от людей, у которых есть потребности, к людям, которые умеют создавать решения, а затем к тем, кто может реализовать и проверить эти решения. Опытный бизнес-аналитик выберет самый эффективный способ доведения любого типа требований до любой аудитории.

Итог разработки требований — задокументированное соглашение между заинтересованными лицами о создаваемом продукте. Как уже говорилось в предыдущих главах, в документе об образе и границах проекта содержатся бизнес-требования, а пользовательские требования часто фиксируются в виде вариантов использования продукта и пользовательских историй. Подробные функциональные и нефункциональные требования к продукту записаны в спецификации к требованиям к ПО, которая предоставляется тем, кто должен проектировать, разрабатывать и проверять решение. Фиксация всех требований вместе в виде структурированного и читабельного материала,

который могут проверить все заинтересованные лица гарантирует, что они понимают, на что соглашаются.

В этой главе рассматриваются цель, структура и содержание спецификации требований к ПО. Мы опишем спецификацию требований к ПО как документ, но это не обязательно должен быть традиционный документ, созданный в обычном редакторе. Надо сказать, что документы обладают многими недостатками:

- сложно хранить описательные атрибуты вместе с требованиями;
- управление изменениями выполняется неуклюже;
- сложно сохранять хронологию версий требований;
- непросто выделить часть требований, которые назначены на определенную итерацию, или отслеживать те требования, что были однажды одобрены, но затем отложены или отменены;
- сложно отслеживать связи требований с другими артефактами разработки.
- дублирование требования, которое логически подходит для размещения в несколько мест, приводит к проблемам с поддержкой.

В качестве альтернативы можно хранить информацию в электронной таблице (у которой те же ограничения, что и у документа), вики, базе данных или серийном средстве управления требованиями (см. главу 30). Их надо рассматривать как разные возможные хранилища или контейнеры информации требований. Но независимо от формы используемого хранилища требований, вам всегда нужны одни и те же типы информации. Описанный в этой главе шаблон спецификации требований к ПО — удобное средство напоминания о том, какую информацию надо собрать и как ее организовать.

Не все считают, что стоит тратить время на документирование требований. А в исследовательских и очень изменчивых проектах, в которых неочевидно, каким будет решение, попытки отслеживать в документации все подробности требований приносит мало пользы. Однако затраты на документирование знаний малы по сравнению со стоимостью получения этих знаний или воссоздания их в какой-то момент в будущем. Акт спецификации и моделирования помогает участникам проекта обдумывать и точно формулировать важные вещи, которые в устном обсуждении могут остаться невыясненными. Если вы *на все сто процентов уверены*, что какая-то информация никому из заинтересованных лиц никогда не будет нужна за исключение времени, когда эта информация хранится в их краткосрочной памяти, тогда можете не регистрировать эту информацию. В противном случае сохраните ее там, где храните что-то типа групповой памяти.

Вам никогда не создать идеальных требований. Помните, что требования пишутся для определенной аудитории. Объем подробностей, типы предоставляемой вами информации и способ ее структурирования должны соответствовать потребностям целевой аудитории. Аналитики естественным образом пишут требования со своей точки зрения, но они должны их писать так, чтобы требования были максимально понятными тем, кто их должен

понимать и выполнять на их основе свою работу. Вот почему важно, чтобы представители этих аудиторий рецензировали требования, проверяя, соответствуют ли требования их потребностям.

Последовательное уточнение деталей — ключевой принцип эффективной разработки требований. В большинстве проектов нереально, да и не нужно, на ранних этапах проект собрать все и каждое требование. Наоборот — надо думать в терминах слоев. Нужно узнать о требованиях достаточно, чтобы можно было в первом приближении определить их приоритеты и назначить на конкретные будущие выпуски и итерации. После этого можно уточнять группы требований в режиме «just in time», предоставив разработчикам достаточно информации, чтобы они не выполняли лишних и ненужных переделок.

Не стоит надеяться, что даже самая точная документация по требованиям сможет заменить текущие обсуждения в процессе проекта. Должны быть открыты все каналы коммуникации: между бизнес-аналитиками, командой разработчиков, представителями клиента и другими заинтересованными лицами, чтобы они могли быстро разобраться с мириадами возникающих по ходу проекта вопросов.

> **Внимание!** Не стоит рассчитывать, что телепатия и ясновидение заменят собой надежные приемы разработки спецификаций требований. Они попросту не работают, хотя некоторые упорно пытаются использовать в качестве основы проектов разработки ПО.

Способов представления требований несколько:

- документация, в которой используется четко структурированный и аккуратно используемый естественный язык;
- графические модели, иллюстрирующие процессы преобразования, состояния системы и их изменения, отношения данных, а также логические потоки и т. п.;
- формальные спецификации, где требования определены с помощью математически точных, формальных логических языков.

Последний метод обеспечивает наивысшую степень точности, однако немногие разработчики — и еще меньше клиентов — знакомы с ним. В большинстве проектов не требуется такого уровня формализма, но я сильно надеюсь, что проектировщики высокорисковых систем, таких как системы управления АЭС, применяют формальные методы спецификации. Структурированный естественный язык, усиленный визуальными моделями и другими приемами представления (такими, как таблицы, рабочие модели, фотографии и математические выражения), остается самым практичным способом документирования требований в большинстве проектов по разработке ПО. Оставшаяся часть главы рассказывает, как можно организовать информацию в спецификации требований к ПО. В главе 11 описываются характеристики высококачественных требований и содержится много рекомендаций по их написанию.

Спецификация требований к ПО

Спецификацию требований в различных компаниях называют по-разному, при этом в этих компаниях названия не используются одинаково. Спецификацию требований к ПО иногда называют *документом бизнес-требований*, *функциональной спецификацией*, *спецификацией продукта* или просто *документом о требованиях*. Так как «спецификация требований к ПО» является отраслевым стандартом, мы называем ее именно так в этой книге (ISO/IEC/IEEE 2011).

В спецификации требований к ПО указываются функции и возможности, которыми должно обладать ПО, а также необходимые ограничения. Она должна содержать достаточно подробное описание поведения системы при различных условиях, а также необходимые качества системы, такие как производительность, безопасность и удобство использования. Спецификация требований служит основой для дальнейшего планирования, дизайна и кодирования, а также базой для тестирования пользовательской документации. Однако она не должна содержать подробности дизайна, проектирования, тестирования и управления проектом за исключением известных ограничений дизайна и реализации. Даже тем, кто работает над проектами гибкой разработки нужна та или иная информация, содержащаяся в хорошей спецификации требований к ПО. Обычно такие разработчики не собирают всю эту информацию в виде целостного материала, но шаблон спецификаций требований служит удобным напоминанием, какие знания может потребоваться собрать. Завершается эта глава разделом, описывающим спецификацию требований в проектах гибкой разработки.

> **Внимание!** Один материал требований часто не удовлетворяет потребностей всех аудиторий. Одним нужно знать только бизнес-цели, другие хотят получить высокоуровневую общую картину, третьих интересует только видение с точки зрения пользователя, но есть и такие, которым требуются все подробности. Вот почему мы активно рекомендуем создавать такие материалы, как документ концепции и границ проекта, документ пользовательских требований и спецификацию требований к ПО. Не следует ожидать, что все представители пользователей прочтут подробную спецификацию требований к ПО и разработчики узнают все, что им нужно, из набора вариантов использования или пользовательских историй.

Спецификация требований к ПО необходима различным участникам проекта:

- клиенты, отдел маркетинга и специалисты по продажам хотят иметь представление о конечном продукте;
- менеджеры проекта по данным спецификации рассчитывают графики, затраты и ресурсы;
- команда разработчиков ПО получает представление о том, какой продукт надо создавать;
- тестировщики составляют основанные на требованиях тесты, планы тестирования и процедуры;

- специалисты по обслуживанию и поддержке получают представление о функциональности каждой составной части продукта;
- составители документации создают руководства для пользователей и окна справки на основании спецификации требований к ПО и дизайна пользовательского интерфейса;
- специалистам по обучению спецификация требований к ПО и пользовательская документация необходима для разработки обучающих материалов;
- персонал, занимающийся юридической стороной проекта, проверяет, соответствуют ли требования к продукту существующим законам и постановлениям;
- субподрядчики строят свою работу и могут нести юридическую ответственность также согласно спецификации требований к ПО.

Если нужной функциональности или качества нет в соглашении о требованиях, нет оснований ожидать, что они появятся в продукте.

Сколько нужно спецификаций?

В большинстве проектов создают одну спецификацию требований к ПО, но в крупных проектах это непрактично. В проектах крупных систем часто пишут спецификацию требований системы, к которой прилагаются отдельные спецификации требований к ПО и оборудованию (ISO/IEC/IEEE, 2011). Одна компания создавала очень сложное приложение управления процессами, в которых на протяжении уже многих лет задействованы более сотни сотрудников. В спецификации требований этого проекта было примерно 800 высокоуровневых требований. Проект был разбит на 20 подпроектов, в каждом из которых были собственные спецификации требований к ПО с примерно 800 или 900 требованиями, выведенными из системных требований. Это большой объем документации, но большие проекты становятся неуправляемыми, если в них не применять подход «разделяй и властвуй».

Есть и другая крайность: в одной компании создали лишь по одному руководящему документу для каждого проекта среднего размера, и назвали его просто — «Спецификация». Она содержала всю возможную информацию о проекте: требования, оценки, проектные планы, планы качества, планы тестирования — в общем, все. Управление изменениями и версиями такого всеобъемлющего документа представляло собой сущий кошмар. Кроме того, уровень информации в таком универсальном документе не соответствовал всем аудиториям, до которых нужно было довести информацию требований.

В третьей компании, где стали применять приемы гибкой разработки, вообще отказались от написания какой-либо формальной документации, — они стали писать пользовательские истории на записках-наклейках. К несчастью для одного проекта клей на записках высох и спустя несколь-

ко месяцев работы над проектом записки часто осыпались на пол, когда кто-то проходил мимо.

В еще одной компании выбрали промежуточный вариант. Хотя их проекты и не были большими и их можно было описать на 40–60 страницах, некоторые члены команды захотели разбить спецификацию требований к ПО на 12 отдельных документов: одну спецификацию для пакетного процесса, одну — для подсистемы отчетности и по одной для каждого из десяти отчетов. Подобный взрывообразный рост числа документов создает проблемы, потому что очень сложно обеспечить синхронизацию и эффективное предоставление нужной информации соответствующим людям.

Лучшим вариантом во всех этих ситуациях будет хранение требований в средстве управления требованиями, как описано в главе 30. Такое средство также оказывается очень кстати для решения дилеммы: создавать одну или несколько спецификаций требований в проекте, где планируется несколько выпусков продукта или итераций разработки (Wiegers, 2006). В этом случае спецификация требований к ПО для части продукта или определенной итерации представляет собой всего лишь отчет, созданный на основе запроса определенного содержимого базы данных.

Не обязательно составлять спецификацию для всего продукта еще до начала разработки, но необходимо зафиксировать требования для каждой итерации перед ее созданием. Это удобно, если в начале работы участники проекта не смогли определить все требования, а некоторую функциональность необходимо быстро передать в руки пользователей. Отклики об использовании ранних итераций позволяют скорректировать оставшуюся часть проекта. Однако при работе над любым проектом необходимо достичь базового соглашения по каждому набору требований до начала их реализации разработчиками. *Выработка базового соглашения* или базовой версии требований (baselining) представляет собой процесс преобразования реализуемых требований к ПО в то, что будет изучаться и одобряться. При работе с согласованным набором требований снижается вероятность недопонимания и ненужных переделок. Подробнее о базовом соглашении о требованиях см. главы 2 и 27.

Структурируйте и составляйте спецификацию требований к ПО таким образом, чтобы все заинтересованные в проекте лица смогли в ней разобраться. Ниже приведены советы, как сделать требования ясными и понятными:

- для структурирования всей необходимой информации используйте соответствующий шаблон;
- разделы, подразделы и отдельные требования должны быть именоваться единообразно;
- используйте средства визуального выделения (такие, как полужирное начертание, подчеркивание, курсив и различные шрифты) последовательно и в разумных пределах; Помните, что цветовые выделения могут быть невидны дальтоникам или при черно-белой печати;

- создайте оглавление, чтобы облегчить пользователям поиск необходимой информации;
- пронумеруйте все рисунки и таблицы, озаглавьте их и, ссылаясь на них, используйте присвоенные номера;
- если при хранении требований в документе вы ссылаетесь в документе на другие его части, используйте возможности работы с перекрестными ссылками в вашем редакторе, а не сложную кодировку страниц;
- если вы используете документы, применяйте гиперссылки, чтобы читатель смог быстро перейти к соответствующим разделам спецификации или другим файлам;
- при хранении требований в специализированном средстве используйте ссылки, чтобы облегчить читателю навигацию по нужной информации;
- включайте графической представление информации, где это возможно для облегчения понимания;
- воспользуйтесь услугами опытного редактора, чтобы обеспечить последовательность документа и единообразие терминологии и структуры.

Требования к именованию

У каждого требования должен быть уникальный и неизменный идентификатор. Это позволит вам ссылаться на определенные требования в запросе на изменения, в хронологии изменений, в перекрестных ссылках или матрице связей требований. При этом также упрощается повторное использование требований в нескольких проектах. Уникальная идентификация требований упрощает взаимодействие между членами команды при обсуждении требований на встрече по взаиморецензированию требований. Простые нумерованные или маркированные списки для этих целей не годятся. Давайте взглянем на достоинства и недостатки некоторых методов именования требований. Выберите любой метод, наиболее подходящий вам.

Номер восемь, с кружочком

Как-то в одном длинном полете на самолете я разговорился с соседом по имени Дейв, который, как оказалось, тоже имеет отношение к разработке ПО. Я упомянул, что меня интересуют требования. Дейв вытащил из портфеля спецификацию требований к ПО — я уж не знаю, носил ли он ее на всякий случай или она оказалась при нем случайно. Я увидел, что требования в этом документе организованы в виде иерархии, но все списки был маркированными. В некоторых местах глубина иерархии достигала восьми уровней. В списках разных уровней использовались разные маркеры — ○, ■, ♦, ✓, ❑, ➪ и т. п. , — но у них не было никаких более содержательных меток кроме этих маркеров. Было невозможно сослаться на элемент маркированного списка или отследить его соответствие в дизайне, сегменте кода или тесте.

Нумерация по порядку

Самый простой способ — присвоить каждому требованию уникальный порядковый номер, например UR-9 или FR-26. Серийные средства управления требованиями присваивают такой идентификатор, когда пользователь добавляет новое требование в базу данных. Префикс обозначает тип требования, например FR означает «functional requirement» (то есть «функциональное требование»). При удалении требования номер повторно не используется, поэтому читателю не придется путаться между исходным требованием FR-26 или новым требование FR-26. Такой простой поход нумерации не обеспечивает логического или иерархического группирования связанных требований, число не подразумевает никакого упорядочения, а названия не раскрывают содержания требования. Вместе с тем это позволяет сохранять уникальные идентификаторы при перемещении требований внутри документа.

Иерархическая нумерация

Это наиболее распространенный способ. Если функциональные требования приводятся в разделе 3.2 спецификации, то все их номера будут начинаться с 3.2. Чем больше цифр, тем больше уровень детализации требования, так что сразу понятно, что 3.2.4.3 является потомком по отношению к 3.2.4. Это способ отличается простотой, компактностью и очевидностью. Ваш текстовый редактор скорее всего сможет назначать номера автоматически. Средства управления требованиями обычно также поддерживают иерархическую нумерацию.

Однако с иерархической нумерацией тоже не все безоблачно. Даже в спецификации среднего размера нумерация может быть весьма длинной. Вдобавок названия, состоящие только из цифр, ничего не говорят о назначении требований. При использовании текстового редактора этот метод не обеспечивает неизменность номеров. Если вы вставите новое положение, то номера всех последующих фрагментов увеличатся. А после удаления или перемещения требования — уменьшатся. Удаление, вставка, слияние или перемещение целых разделов приводит к изменению многих фрагментов. При этом нарушаются все ссылки на эти фрагменты.

> **Внимание!** Однажды один бизнес-аналитик заметил: «Мы не разрешаем людям вставлять требования — сбивается нумерация». Не позволяйте применять неэффективные приемы, которые могут препятствовать эффективной и разумной работе.

Можно усовершенствовать этот способ, пронумеровав важнейшие разделы требований иерархически, а затем выделив отдельные функциональные требования в каждом разделе с помощью короткого текстового кода, после которого следует порядковый номер. Например, если в спецификации требований к ПО есть «Раздел 3.5 — Функции редактора», то требования в нем будут названы ФР-1, ФР-2 и т. д. При этом соблюдается определенная иерархия и структурированность документа, названия достаточно короткие,

осмысленные и менее зависят от местоположения в документе. Но это не решает полностью проблему нумерации.

Иерархические текстовые теги

Консультант Том Гилб (Gilb, 1988) предложил текстовую схему иерархических тегов именования требований. Рассмотрим такое требование: «Система должна запрашивать у пользователя подтверждение запроса, когда тот хочет печатать более 10 копий». Этому требованию можно присвоить тег Print.ConfirmCopies, который означает, что это требование является частью функции печати и связано с количеством печатаемых копий. Иерархические текстовые тэги структурированы, их названия осмысленны и не меняются при добавлении, удалении или перемещении остальных требований. Пример спецификации требований к ПО в Приложении B иллюстрирует эту схему, как другие примеры в тексте книги. Этот метод также подходит для нумерации бизнес-правил, если они управляются вручную, а не с помощью специализированного инструмента или хранилища бизнес-правил.

Использование таких текстовых тегов помогает решить еще одну проблему. При иерархической организации между требованиями возникают отношения «родитель–потомок». Если родитель описан, как функциональное требование, отношение между потомком и родителем может быть непонятным. Правильное соглашение предусматривает, что родительское требование должно выглядеть как заголовок или название функции, а не как самостоятельное функциональное требование. Потомки родительского требования в совокупности должны покрывать все его возможности. Вот пример, содержащий заголовок и четыре функциональных требования.

Продукт:	Заказ товаров в интернет-магазине
.Cart	В интернет-магазине должна быть корзина, в которой размещаются все выбранные клиентом товары
.Discount	В корзине должно присутствовать поле для кода скидки. Код скидки принимает вид определенного процента на все содержимое корзины или фиксированную долларовую скидку на конкретные товары в корзине
.Error	При вводе клиентом неверного кода скидки интернет-магазин должен отображать сообщение об ошибке
.Shipping	Корзина должна прибавлять к заказам клиентов стоимость доставки, если клиент заказывает физические товары, которые нужно доставлять

Полный уникальный идентификатор каждого требования создается путем присоединения метки требования к строке, состоящей из меток вышестоящих родителей. Утверждение **Product** выглядит как заголовок, а не как конкретное требование. Первое функциональное требование обозначено как **Product.Cart**. Полный идентификатор третьего требования — **Product.**

Discount.Error. Такая иерархическая схема избавляет от проблем с обслуживанием иерархического нумерования, но теги длинные и им нужно назначать значащие имена, обычно на основе имени соответствующей функции. Может быть сложным обеспечить уникальность, особенно если над набором требований трудится несколько человек. В небольших наборах требований можно упростить схему, совместив иерархическую нумерацию с суффиксом порядкового номера: *Product.Cart.01*, *Product.Cart.02* и т. д. В общем, существует много рабочих схем.

Когда информации недостаточно

Иногда вы вдруг понимаете, что вам не хватает определенной информации о конкретном требовании. Чтобы пометить пробелы в данных, используйте пометку «TBD» (to be determined — необходимо определить). Выясните все записи TBD до реализации набора требований. Любые неясности повышают риск ошибки со стороны разработчика или тестировщика и, как следствие, вероятность переделки. Если разработчику не хватает информации, он не всегда обращается для разрешения проблемы к источнику требования. Иногда он пытается снять вопрос самостоятельно, реализуя лучший с его точки зрения вариант, который не всегда корректен для проекта в целом. Если же необходимо приступить к созданию продукта, а неразрешенные вопросы еще остаются, следует или отложить реализацию неясных требований, или сделать эти части продукта легко модифицируемыми, чтобы позже разрешить оставшиеся открытыми проблемы. Запишите все пробелы и другие вопросы по требованиям в списке проблем. По мере снижения числа открытых проблем будут стабилизироваться требования. Подробнее об управлении и разрешении открытых проблем см. главу 27.

> **Внимание!** Неясности не решаются сами собой. Пронумеруйте все пометки «TBD», запишите, кто отвечает за решение вопроса и в какой срок, регулярно отслеживайте их состояние вплоть до разрешения.

Пользовательские интерфейсы и спецификация требований к ПО

Включение элементов пользовательского интерфейса в спецификацию имеет как преимущества, так и недостатки. Положительным моментом можно считать то, что бумажные прототипы, рабочие модели, каркасы или средства моделирования делают требования осязаемыми как для пользователей, так и разработчиков. Как говорится в главе 15, существуют мощные приемы выявления и проверки требований. Если у пользователей продукта есть ожидания насчет того, как должны выглядеть и вести себя те или иные части продукта, а, значит, они будут разочарованы отсутствием реализации своих ожиданий, эти ожидания относятся к требованиям.

Отрицательным моментом можно считать то, что изображения и архитектура пользовательского интерфейса отображают решения (дизайн), а не тре-

бования. Их включение в спецификацию требований к ПО делает документ тяжеловесным, а большие документы пугают некоторых людей. Откладывая создание базового соглашения решений в спецификации требований к ПО до завершения разработки пользовательского интерфейса, вы можете замедлить разработку и заставить нервничать людей, которые и так обеспокоены тем, что на работу над требованиями затрачено слишком много времени. Включение дизайна пользовательского интерфейса может привести к тому, что визуальный дизайн будет определять требования, что часто ведет к пропуску функций. Те, кто пишет требования, не всегда обладают достаточной квалификацией для создания дизайна пользовательских интерфейсов. Кроме того после того, как заинтересованные лица увидят пользовательский интерфейс в спецификации требований (или где-либо еще), им сложно будет его «забыть». Визуализация на ранних этапах может прояснить требования, но затруднять улучшение пользовательского интерфейса со временем.

Макет экрана не заменит пользовательских и функциональных требований. Не следует ожидать, что разработчики смогут сделать вывод о базовой функциональности и взаимосвязи данных по снимкам экрана. У одной компании, создающей ПО для Интернета, постоянно возникали одни и те же проблемы из-за того, что разработчики непосредственно переходили к работе над визуальным дизайном в считанные часы после подписания контракта. У них не было ясного представления о том, как пользователи будут работать с веб-сайтом, поэтому им приходилось тратить массу времени на последующие исправления.

Если вам действительно нужно реализовать определенную функциональность с применением конкретных элементов управления на пользовательском интерфейсе и конкретной структуры экрана, тогда нужно и важно включить эту информацию в спецификацию требований к ПО в качестве ограничений дизайна. Ограничения дизайна лимитируют возможности дизайнера пользовательского интерфейса. Просто нужно быть уверенным, что ограничения не накладываются необоснованно, слишком рано или по неправильным причинам. Если спецификация требований описывает улучшения существующей системы, часто имеет смысл включить образы экрана в виде, в котором их нужно реализовать. Возможности разработчиков уже ограничены рамками существующей системы, поэтому бывает, что заранее известно, как ее надо менять и как будут выглядеть существующий — и, возможно, новый — интерфейс.

«Золотая середина» подразумевает включение концептуальных изображений выбранных экранов — я их называю набросками, несмотря на качество их выполнения, — в требования без обязательного точного соблюдения этих моделей при реализации. На рис. 10-1 показан пример наброска веб-страницы. Включение таких набросков в спецификацию требований предоставляет еще одно представление требований, при этом ясно говорится, что это наброски, а не готовый вид интерфейсе. Например, предварительный набросок сложного диалогового окна может проиллюстрировать назначение части требований,

однако опытный дизайнер интерфейсов сумеет превратить его в диалоговое окно с вкладками для повышения удобства работы пользователя.

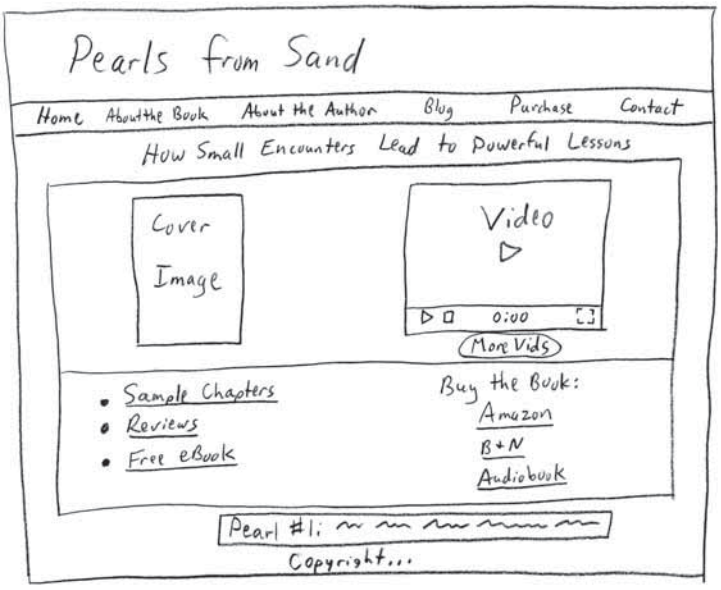


Рис. 10-1. Пример наброска пользовательского интерфейса, подходящего для включения в документ требований

В командах, работающих над проектами со многими окнами и экранами, часто предпочитают документировать особенности дизайна пользовательского интерфейса в отдельной спецификации пользовательского интерфейса или с помощью средства конструирования интерфейс или создания прототипов. Используйте такие приемы, как создание моделей «отображение-действие-реакция», для описания имена элементов на экране, их свойств и их подробного поведения (Beatty и Chen, 2012).

Шаблон спецификации требований к ПО

Каждая организация, специализирующаяся на разработке ПО, должна принять один или несколько стандартных шаблонов спецификации требований к ПО для использования в проектах. Доступны различные шаблоны спецификации (например: ISO/IEC/IEEE 2011; Robertson и Robertson, 2013). Если вы беретесь за проекты различных типов и размеров, от конструирования новой объемной системы до небольших улучшений уже работающих систем, позаботьтесь для проектов каждого крупного класса завести отдельный шаблон спецификации. Подробнее о том, как эффективно использовать шаблоны документов см. врезку «Использование шаблонов» в главе 5.

На рис. 10-2 показан шаблон спецификации требований, который подходит для многих проектов. В Приложении В показан пример спецификации требо-

ваний к ПО, производной от этого шаблона. Этот шаблон вместе с инструкциями по применению в каждом разделе можно загрузить с веб-сайта этой книги. Некоторые люди форматируют инструкции как скрытый текст в Microsoft Word. Это позволяет оставить подсказки в документе. Чтобы увидеть эти инструкции, нужно включить режим отображения непечатных символов.

1. Введение
   1.1 Назначение
   1.2 Соглашения, принятые в документах
   1.3 Границы проекта
   1.4 Ссылки
2. Общее описание
   2.1 Общий взгляд на продукт
   2.2 Классы и характеристики пользователей
   2.3 Операционная среда
   2.4 Ограничения дизайна и реализации
   2.5 Предположения и зависимости
3. Функции системы
   3.x Функция системы X
      3.x.1 Описание
      3.x.2 Функциональные требования
4. Требования к данным
   4.1 Логическая модель данных
   4.2 Словарь данных
   4.3 Отчеты
   4.4 Получение, целостность, хранение и утилизация данных
5. Требования к внешним интерфейсам
   5.1 Пользовательские интерфейсы
   5.2 Интерфейсы ПО
   5.3 Интерфейсы оборудования
   5.4 Коммуникационные интерфейсы
6. Атрибуты качества
   6.1 Удобство использования
   6.2 Производительность
   6.3 Безопасность
   6.4 Техника безопасности
   6.x [Другие]
7. Требования по интернационализации и локализации
8. Остальные требования

Приложение А. Словарь терминов

Приложение Б. Модели анализа

Рис. 10-2. Предлагаемый шаблон для спецификации требований к ПО

Иногда фрагмент информации логически подходит для нескольких разделов шаблона. Выберите один раздел и используйте именно его для информации такого типа в своем проекте. Не дублируйте информацию в нескольких разделах, даже если логически она ложится в эти разделы (Wiegers, 2006). Используйте перекрестные ссылки и гиперссылки, чтобы облегчить читателям поиск нужной информации.

При создании документов требований применяйте эффективные приемы и средства управления версиями, чтобы все читатели четко понимали, какую

версию они читают в тот или иной момент времени. Ведите журнал изменений, в котором фиксируются суть изменений, автор, дата и причина (см. главу 27). В оставшейся части этой главы описывается информация, которая должна присутствовать в отдельных разделах спецификации требований к ПО.

> **Внимание!** Вы можете внедрять материал с помощью ссылки в другие документы, а не дублировать его в спецификации требований к ПО. Для этого можно использовать гиперссылки в качестве ссылок для отслеживания связей, определенных в средстве управления требованиями. С гиперссылками связан тот риск, что они могут «ломаться» при изменении иерархии папок, где хранятся документы. Подробнее о повторном использовании существующих знаний о требованиях см. главу 18.

1. Введение

Введение представляет собой обзор, помогающий читателям разобраться в структуре и принципе использования спецификации требований к ПО.

1.1 Назначение

Определите продукт или приложение, требования для которого указаны в этом документе, в том числе редакцию или номер выпуска. Если эта спецификация требований к ПО относится только к части системы, идентифицируйте эту часть или подсистему. Опишите типы читателей, которым адресован этот документ, например разработчикам, менеджерам проектов, маркетологам, пользователям, тестировщикам или составителям документации.

1.2 Соглашения, принятые в документах

Опишите все стандарты или типографические соглашения, включая значение стилей текста, особенности выделения или нотацию. Если вы нумеруете требования вручную, можно определить принятый формат на случай, если кому-нибудь позже понадобиться добавить требование.

1.3 Границы проекта

Кратко опишите ПО и его назначение. Покажите, как связан продукт с пользователями или корпоративными целями, а также с бизнес-целями и стратегиями. Если имеется отдельный документ о концепции и границах проекта, не повторяйте его содержимое, а просто сошлитесь на него. Если спецификацию требований к ПО предполагается разрабатывать постепенно, она должна содержать собственное положение о концепции и границах продукта в качестве подраздела долгосрочной стратегической концепции. Можно предоставить высокоуровневую сводку главной функциональности выпуска или функций, которые он должен выполнять.

1.4 Ссылки

Перечислите все документы или другие ресурсы, на которые вы ссылаетесь в этой спецификации, в том числе гиперссылки на них, если их местополо-

жение меняться не будет. Это могут быть руководства по стилям пользовательского интерфейса, контракты, стандарты, спецификации к системным требованиям, спецификации интерфейса и спецификации требований к ПО связанных продуктов. Объем информации должен быть достаточным для того, чтобы пользователь сумел при необходимости получить доступ к каждому указанному материалу, а именно: название, имя автора, номер версии, дата, источник, место хранения или URL-адрес.

2. Общее описание

В этом разделе представлен общий обзор продукта и среды, в которой он будет применяться, предполагаемая пользовательская аудитория, а также известные ограничения, предположения и зависимости.

2.1 Общий взгляд на продукт

Опишите контекст и происхождение продукта. Поясните, является он новым членом растущего семейства продуктов, новой версией существующей системы, заменой существующего приложения или совершенно новым продуктом. Если спецификация требований определяет компонент более крупной системы, укажите, как это ПО соотносится со всей системой и определите основные интерфейсы между ними. Неплохо также включить визуальные модели, такие как контекстную диаграмму или карту экосистемы (описаны в главе 5), чтобы показать взаимосвязь продукта с другими системами.

2.2 Классы и характеристики пользователей

Определите различные классы пользователей, которые, как предполагается, будут работать с вашим продуктом, и опишите их соответствующие характеристики (см. главу 6). Некоторые требования могут относиться только к определенным классам пользователей. Определите привилегированные классы пользователей. Классы пользователей представляют подмножество заинтересованных в проекте лиц, их описание приводится в документе концепции и границ проекта. Описания классов пользователей являются повторно используемым ресурсом. Если есть главный каталог классов пользователей, можно включить описания классов пользователей, просто указав их в каталоге, не дублируя информацию.

2.3 Операционная среда

Опишите рабочую среду, в которой будет работать ПО, включая аппаратную платформу, операционные системы и их версии, а также географическое местоположение пользователей, серверов и баз данных вместе с организациями, в которых располагаются соответствующие базы данных, серверы и вебсайты. Перечислите все остальные компоненты ПО или приложения, с которыми система должна быть совместима. Если в связи с разработкой новой системы нужно произвести значительную работу с технической инфраструк-

турой, стоит подумать о создании отдельных требований к инфраструктуре, в которой детально изложить подробности этой работы.

2.4 Ограничения дизайна и реализации

Бывает, что нужно использовать вполне определенный язык программирования, определенную библиотеку, на разработку которой уже потрачено время, и т. п. Опишите все факторы, которые ограничат возможности, доступные разработчикам, и логически обоснуйте каждое положение. Требования, которые включают или написаны в форме идей по решению, а не потребностей накладывают ограничения на дизайн, часто неоправданные, поэтому за этим надо следить. Подробнее об ограничениях см. статью 14.

2.5 Предположения и зависимости

Предположение (assumption) — это утверждение, которое предполагается верным в отсутствие знаний или доказательств иного. Проблемы возможны в том случае, если предположение неверны, устарели, не находятся в совместном использовании или изменяются, поэтому определенные предположения можно отнести к группе рисков проекта. Один читатель спецификации требований к ПО может считать, что продукт будут соответствовать особому стандарту пользовательского интерфейса, тогда как другой предположит нечто совершенно иное. Разработчик может думать, что определенный набор функций написан специально для этого приложения, бизнес-аналитик — что он будет взят из предыдущего проекта, а менеджер проекта — что предполагается приобрести коммерческую библиотеку функций. Включаемые здесь предположения относятся к системной функциональности; предположения относящиеся к бизнесу представлены в документе концепции и границ проекта, как описано в главе 5.

Определите все *зависимости* проекта или создаваемой системы от внешних факторов или компонентов вне ее контроля. Например, до установки продукта может требоваться установить Microsoft .NET Framework 4.5 или более позднюю версию — это зависимость.

3. Функции системы

Шаблон на рис. 10-2 структурирован по функциям системы — это еще один способ систематизации функциональных требований. Другие методы классификации — по функциональным областям, рабочим потокам, вариантам использования, режимам работы, классам пользователей, стимулам и реакциям. Возможны также иерархические комбинации этих элементов, например варианты использования внутри классов пользователей. Не существует единственно правильного метода организации; выберите тот, при котором пользователям будет легче понять предполагаемые возможности продукта. Мы опишем схему функций на примере.

3.x Функция системы X

Опишите название особенности несколькими словами, например «3.1 Проверка правописания». Так же назовите подразделы с 3.x.1 по 3.x.3 для каждой функции системы.

3.x.1 Описание

Кратко опишите функцию системы и укажите ее приоритет: высокий, средний или низкий приоритетом (см. главу 16.) Приоритеты являются динамичной характеристикой, они могут изменяться в ходе проекта. Если вы используете средство управления требованиями, определите атрибут требований для обозначения приоритета. Атрибуты требований обсуждаются в главе 27, а средства управления требованиями — в главе 30.

3.x.2 Функциональные требования

Перечислите по пунктам конкретные функциональные требования, которые связаны с этой функцией. Именно эти функции ПО нужно реализовать, чтобы пользователь мог использовать сервисы этой функции или реализовать вариант использования. Опишите, как продукт должен реагировать на ожидаемые ошибки, неправильный ввод информации или неверные действия. Присвойте каждому функциональному требованию уникальное имя, как описано ранее в этой главе. При использовании средства управления требованиями можно создать много атрибутов для каждого функционального требования, таких как основание, источник и состояние.

4. Требования к данным

Ценность информационных систем заключается в том, что они предоставляют возможность манипулировать данными. Используйте этот раздел шаблона для описания различных аспектов данных, которые будет потреблять система в качестве входной информации, как-то обрабатывать и возвращать в виде выходной информации. Подробнее см. главу 13. Стивен Уитхолл (Stephen Withall, 2007) описывает много схем для точного документирования данных (которые также называют информацией).

4.1 Логическая модель данных

Как говорится в главе 13, модель данных это визуальное представление объектов и наборов данных, которые будет обрабатывать система, а также отношений между ними. Существует много видов нотации для моделирования данных, в том числе диаграммы отношений «сущность–связь» и диаграммы классов UML. Можно включить модель данных для бизнес-операций, выполняемых системой или логическое представление данных, с которыми будет работать система. Это не то же самое, что модель данных реализации, которая реализуется в виде дизайна базы данных.

4.2 Словарь данных

Словарь данных определяет состав структур данных, а также их значение, тип данных, длину, формат и разрешенные значения элементов данных, из которых состоят эти структуры. Серийные средства моделирования данных часто включают компонент-словарь данных. Во многих случаях словарь данных лучше хранить как отдельный артефакт, не внедряя его в спецификацию требований к ПО. Это повышает возможности повторного использования в других проектах. Словарь данных обсуждается в главе 13.

4.3 Отчеты

Если приложение будет генерировать отчеты, перечислите их здесь и опишите их характеристики. Если отчет должен соответствовать определенному готовому макету, можно указать это как ограничение, возможно с примером. В противном случае сосредоточьтесь на логических описаниях, порядке сортировки, уровнях суммирования и т. п., отложив подробный макет до этапа дизайна. Подробнее об определении отчетов — в главе 13.

4.4 Получение, целостность, хранение и утилизация данных

Если это важно, опишите, как получают и обслуживают данные. Например, при открытии канала номенклатуры данных может требоваться первым делом выполнить начальный дамп всей номенклатуры данных в принимающую систему, а после этого использовать каналы для передачи только изменений. Укажите все требования, относящиеся к защите целостности данных системы. Укажите все процедуры, которые могут потребоваться, например резервное копирование, создание контрольных точек, зеркальное отображение или проверка корректности данных. Укажите политики, которые должна применять система для хранения или утилизации данных, в том числе временных данных, метаданных, остаточных данных (таких как удаленные записи), данных в кеше, локальных копий, архивов и промежуточных архивов.

5. Требования к внешним интерфейсам

В этом разделе указывается информация, которая гарантирует, что система будет правильно взаимодействовать с пользователями и компонентам внешнего оборудования и ПО. Выработка согласия по внешнему и внутреннему интерфейсу системы признано оптимальным приемом в области разработки ПО (Brown, 1996). В сложной системе с множеством подкомпонентов следует использовать раздельные спецификации для интерфейсов или спецификацию системной архитектуры. В документацию по интерфейсу можно включить ссылки на материал из других документов. Например, ссылка может указать на руководство по работе с устройством, где перечислены коды ошибок, которые устройство может отправить программе.

Войны интерфейсов

Две команды разработчиков ПО объединились для создания флагманского продукта компании A. Datum Corporation. Команда, отвечающая за базу знаний, создала сложное ядро анализа на C++, а команда, отвечающая за приложения, реализовала пользовательский интерфейс на Java. Подсистемы взаимодействовали между собой посредством API. К сожалению, команда, отвечающая за базу знаний, периодически модифицировала API, в результате чего систему не удавалось собрать и запустить на выполнение должным образом. Команде, отвечающей за приложения, требовалось несколько часов, чтобы распознать все проблемы и определить основную причину — изменение API. Эти изменения не согласовывались, не доводились до сведения всех заинтересованных в проекте лиц и не были скоординированы с соответствующими модификациями в коде на Java. Изменение интерфейса *обязательно* требует уведомления об этом людей, группы или системы на другой стороне этого интерфейса. Интерфейс скрепляет компоненты вашей системы — включая пользователей, поэтому необходимо документировать детали интерфейса и синхронизировать модификации в процессе управления изменениями в проекте.

5.1 Пользовательские интерфейсы

Опишите логические характеристики каждого пользовательского интерфейса, который необходим системе. Некоторые особенные характеристики пользовательских интерфейсов могут упоминаться в разделе «6.1 Удобство использования». Некоторые из них перечислены здесь:

- ссылки на стандарты графического интерфейса пользователей или стилевые рекомендации для семейства продуктов, которые необходимо соблюдать;
- стандарты шрифтов, значков, названий кнопок, изображений, цветовых схем, последовательностей полей вкладок, часто используемых элементов управления, графики фирменного стиля, уведомления о зарегистрированных товарных знаках и о конфиденциальности и т.п.;
- размер и конфигурация экрана или ограничения разрешения;
- стандартные кнопки, функции или ссылки перемещения, одинаковые для всех экранов, например кнопка справки;
- сочетания клавиш;
- стандарты отображения и текста сообщений;
- стандарты проверки данных (такие как ограничения на вводимые значения и когда нужно проверять содержимое полей);
- стандарты конфигурации интерфейса для упрощения локализации ПО;
- специальные возможности для пользователей с проблемами со зрением, различением цвета и другими ограничениями.

5.2 Интерфейсы ПО

Опишите связи продукта и других компонентов ПО (идентифицированные по имени и версии), в том числе другие приложения, базы данных, операционные системы, средства, библиотеки, веб-сайты и интегрированные серийные компоненты. Укажите назначение, форматы и содержимое сообщений, данных и контрольных значений, обмен которыми происходит между компонентами ПО. Опишите соответствия между входными и выходными данными между системами и все преобразования, которые должны происходить с данными при перемещении между системами. Опишите службы, необходимые внешним компонентам ПО, и природу взаимодействия между компонентами. Определите данные, которыми будут обмениваться и к которым будут иметь общий доступ компоненты ПО. Определите нефункциональные требования, влияющие на интерфейс, такие как уровни обслуживания для времени и частоты отклика или меры и ограничения безопасности. Часть этой информации может быть определена как требования к данным в разделе 4 или как требования к взаимодействию в разделе «6. Атрибуты качества».

5.3 Интерфейсы оборудования

Опишите характеристики каждого интерфейса между компонентами ПО и оборудования системы. В описание могут входить типы поддерживаемых устройств, взаимодействия данных и элементов управлений между ПО и оборудованием, а также протоколы взаимодействия, которые будут использоваться. Перечислите входные и выходные данные, их формат, разрешенные значения или их диапазоны, а также все временны*е* характеристики, о которых должны знать разработчики. Если такой информации очень много, лучше создать отдельный документ спецификации интерфейса. Подробнее об определении требований к системам, содержащим аппаратные устройства, см. главу 26.

5.4 Коммуникационные интерфейсы

Укажите требования для любых функций взаимодействия, которые будут использоваться продуктом, включая электронную почту, веб-браузер, сетевые протоколы и электронные формы. Определите соответствующие форматы сообщений. Опишите особенности безопасности взаимодействия или шифрования, скорости передачи данных и механизмов согласования и синхронизации. Укажите все ограничения этих интерфейсов, например допустимость тех или иных типов вложений в сообщениях электронной почты.

6. Атрибуты качества

В этом разделе описываются нефункциональные требования помимо ограничений, описанных в разделе 2.4, и требований к внешним интерфейсам, описанным в разделе 5. Эти характеристики должны быть точно определены и поддаваться проверке и измерению. Укажите относительные приори-

теты различных атрибутов, например приоритет простоты использования над легкостью изучения или приоритет безопасности над производительностью. Необходимые степени качества удается гораздо эффективнее описать с помощью подробных нотаций спецификации, таких, как Planguage, чем с помощью простых описательных утверждений (см. главу 14). Подробнее о требованиях к атрибутам качества и примерах см. главу 14.

6.1 Удобство использования

Требования к удобству использования подразумевают легкость изучения, простоту использования, предотвращение ошибок и восстановление, эффективности взаимодействия и специальные возможности. Указанные в этом разделе требования к удобству использования помогут дизайнеру интерфейсов создать максимально удобную для пользователя рабочую среду.

6.2 Производительность

Укажите конкретные требования к производительности для различных системных операций. Если у различных функциональных требований или функций имеются разные требования к производительности, то следует указывать задачи, связанные с производительностью, там же, в разделе соответствующих функциональных требований, а не включать их все в этот раздел.

6.3 Безопасность

Укажите все требования, касающиеся безопасности или конфиденциальности, которые ограничивают доступ или возможности использования продукта. Это может быть физическая безопасность, а также защита данных или ПО. Источником требований к безопасности, как правило, являются бизнес-правила, поэтому определите политики или положения, касающиеся защиты или конфиденциальности, которым продукт должен соответствовать. Если они задокументированы и хранилище бизнес-правил, просто сошлитесь на них.

6.4 Техника безопасности

В этом разделе укажите требования, связанные с возможными потерями, повреждениями или ущербом, которые могут быть результатом использования продукта. Определите меры безопасности или упреждающие действия, которые можно предпринять, так же как и потенциально опасные действия, которые можно предотвратить. Определите сертификаты по безопасности, политики или положения, которым продукт должен соответствовать.

6.x [Другие]

Создайте в спецификации требований к ПО отдельный раздел для каждого дополнительного атрибута качества продукта, чтобы описать характеристики, которые будут важны для клиентов или для разработчиков и людей, от-

ветственных за поддержку. Это может быть доступность, возможность установки, целостность, возможность модификации, переносимость, надежность, устойчивость, масштабируемость и контролируемость. В главе 14 описывается процедура работы с такими атрибутами, которые очень важны в конкретном проекте.

7. Требования по интернационализации и локализации

Требования по интернационализации и локализации обеспечивают возможность использовать продукт в других странах, региональных стандартах и географических районах, отличающихся от тех, в которых он был создан. Такие требования могут быть направлены на разрешение различий в валютах, форматировании дат, чисел, адресов и телефонных номеров, языках, в том числе различных вариантах одного языка (например, американского и британского вариантов английского), используемых символах и наборах символов, личных именах и фамилиях, часовых поясах, международных нормативных актах и законах, культурных и политических традициях, размере используемой бумаги, единицах веса и меры, электрическом напряжении и конфигурации электрических разъемов и во многом другом. Требования по интернационализации и локализации вполне могут повторно использоваться во многих проектах.

8. [Остальные требования]

Определите все другие требования, которые еще не были описаны в спецификации требований к ПО. Примером могут служить юридические, законодательные или финансовые требования и требования стандартов, требования к установке, конфигурированию, запуску и остановке продукта, а также к журналированию, мониторингу и контрольному следую. Вместо того чтобы сливать всю эту информацию в один раздел, добавьте любые новые разделы к шаблону вашего проекта. Пропустите этот раздел, если все необходимые требования уже расписаны в других разделах. В этот раздел можно включить требования к переходу, которые необходимы для миграции с предыдущей системы на новую, если они относятся к создаваемому ПО (например, программы преобразования данных), в противном случае их можно включить в план управления проектом (например, разработка обучающих материалов или поставка).

Приложение А. Словарь терминов

Определите все специальные термины, которые читателю необходимо знать для правильного понимания спецификации требований к ПО, включая сокращения и аббревиатуры. Расшифруйте каждое сокращение и приведите его определение. Подумайте о создании расширенного общекорпоративного словаря для нескольких проектов, который включает по ссылке все термины, относящиеся к данному проекту. В этом случае в спецификации требований

к ПО будут определены только те термины, которые относятся лишь к данному проекту и которых нет в общекорпоративном словаре. Заметьте, что определения данных находятся в словаре данных, а не терминов.

Приложение Б. Модели анализа

В этом необязательном разделе описывается, а точнее напоминается, о таких моделях анализа, как диаграммы потоков данных, деревья функций, диаграммы переходов состояния и диаграммы «сущность-связь» (см. главу 12). Часто читателю удобнее, когда определенные модели внедрены в соответствующие разделы спецификации, а не собраны скопом в конце.

Спецификация требований в проектах гибкой разработки

В проектах гибкой разработки (agile) применяется ряд способов определения требований, которые отличаются от только что описанного метода. Как вы видели в главе 8, во многих проектах гибкой разработки в процессе выявления требований используются пользовательские истории. Каждая такая история представляет собой формулировку потребности пользователя или функциональности, которая может быть полезной пользователю или покупателю системы (Cohn, 2004; Cohn, 2010). В проектах гибкой разработки команда может начать спецификацию, записав достаточно информации по каждой пользовательской истории, чтобы заинтересованные лица получили общее представление, о чем история, и смогли определить приоритеты историй друг относительно друга. Это также позволяет команде начать планировать назначение историй на конкретные итерации. Команда может собирать связанные истории в «минимально поддающуюся для продвижения на рынке функцию», которую нужно реализовать целиком к выпуску продукта, чтобы клиент мог получить от нее пользу.

Пользовательские истории накапливаются и приоритизируются в *резерве продукта* (product backlog), который меняется на протяжении всего проекта. Крупные истории, которые охватывают значительную функциональность и которые нельзя реализовать в одной итерации, разбиваются на более мелкие, которые назначаются конкретным итерациям (см. главу 20). Пользовательские истории можно записывать на чем-то простом, например на карточках каталога, а не в традиционном документе. Одни команды гибкой разработки сохраняют свои истории в средстве управления историями, другие после реализации их не сохраняют.

Когда команда приступает к очередной итерации, каждая история, назначенная на эту итерацию, уточняется и наполняется деталями в процессе обсуждения между владельцем продукта, людьми, выполняющими роль бизнес-аналитика, аналитиками, тестировщиками и пользователями. То есть спецификация подразумевает постепенное уточнение подробностей на

правильном этапе проекта, что является хорошей практикой в любом проекте. Обычно эти подробности соответствуют тому, что мы называли функциональными требованиями в спецификации требований к ПО. Однако в проектах гибкой разработки эти подробности часто представляются в форме приемочных тестов, описывающих, как система должна вести себя при правильной реализации истории. Тесты истории проводятся во время итерации, в которой реализуется эта история, и в будущих итерациях в рамках регрессионного тестирования. Как и в любых других тестах, в них должны выполняться исключительные условия, а также ожидаемое поведение. Эти приемочные тесты могут записываться на карточках или регистрироваться в более постоянной форме, например в средстве тестирования. Тесты нужно автоматизировать, чтобы обеспечить быстрое и полное регрессионное тестирование. Если команда решит отказаться от исходных пользовательских историй, тогда приемочными тестами может быть постоянная документация требований, если они хранятся в специализированном средстве.

Аналогично нефункциональные требования могут записываться на карточках, но не как пользовательские истории, а как ограничения (Cohn, 2004). Альтернативный вариант — указывать нефункциональные требования, относящиеся к определенной пользовательской истории, в форме критериев приемки как демонстрацию достижения определенного атрибута качества. Например, тесты безопасности могут демонстрировать, что только определенным пользователям разрешен доступ к функциональности, описанной в соответствующей пользовательской истории, а остальным пользователям доступ закрыт. Команде гибкой разработки не возбраняется применять другие методы представления знаний о требованиях, такие как модели анализа или словарь данных. Они должны выбрать метод представления, который привычен и уместен в их культуре и проект.

Выбор надлежащих форм спецификации требований к ПО — исключительная прерогатива команды проекта. Вспомните главную цель разработки требований — собрать согласованное понимание требований, качество которых *достаточно*, чтобы приступить к разработке следующей части продукта и продолжить работу при приемлемом уровне риска. «Надлежащий» уровень формальности и подробности документа требований зависит от многих факторов, в числе которых:

- объем оперативного неформального вербального и визуального общения между клиентами и разработчиками, который необходим для получения подробностей, необходимых для правильной реализации каждого пользовательского требования;
- предел, до которого неформальные методы общения могут обеспечить эффективную синхронизацию во времени и пространстве внутри команды;
- граница, до которой представляет ценность или необходимость сохранение знаний о требованиях для будущих улучшений, реинжиниринга приложения, проверки, аудита, проверки на соответствие нормативам, сертификации продукта или в соответствии с условиями контракта;

- граница, до которой приемочные тесты могут эффективно заменять описания ожидаемых возможностей и поведения системы;
- предел, до которого человеческая память может заменить письменное представление.

Независимо от типа создаваемого командой продукта, используемой методики разработки или используемых бизнес-аналитиком приемов выявления требований, эффективная спецификация требований жизненно важна для успеха. Существует много путей достижения этой цели. Просто помните, что если не определить высококачественные требования, полученный в результате продукт будет представлять собой коробку с сюрпризами — вы никогда не будете знать, что от него можно ожидать.

Что дальше?

- Проверьте набор требований проекта на предмет соответствия шаблону на рис. 10-2, чтобы определить, собраны ли все требования в разделах, относящихся к вашему проекту. Эта глава скорее не о том, как наполнять определенный шаблон, а больше о том, как гарантировать, что вы собрали всю информацию, необходимую для успеха проекта, — шаблон всего лишь напоминание.
- Если в вашей организации еще нет стандартного шаблона спецификации требований к ПО, соберите небольшую рабочую группу для того, чтобы принять такой шаблон. Начните с шаблона на рис. 10-2 и измените его так, чтоб он наилучшим образом соответствовал потребностям ваших проектов и продуктов. Выработайте стандарт именования отдельных требований.
- Если вы храните требования в форме, отличной от обычного документа, например в средстве управления требованиями, изучите шаблон спецификации требований к ПО на рис. 10-2 и определите, нет ли каких-либо категорий информации требований, которые вы сейчас не выявляете и не фиксируете. Измените свое хранилище, включив эти категории, чтобы в будущем хранилище напоминало о них при выявлении требований.

Глава 11

Пишем идеальные требования

«Привет, Гаутам. Это звонит Рут из филиала в Остине. Мы получили последнюю версию ПО веб-сайта для музыкального интернет-магазина. У меня вопрос о функциональности предварительного прослушивания песен. Она работает не так, как я себе представляла».

«Давай я посмотрю требования, которые ты присылала, — ответил Гуатам. — А, вот они. В пользовательской истории говорится: «Как клиент я хочу прослушать фрагменты песен, чтобы решить, какую приобрести». В моих заметках говорится, что когда мы это обсуждали, ты сказала, что длительность фрагментов песен должна быть 30 секунд и что должен использоваться наш встроенный проигрыватель MP3-плейер, чтобы клиенту не нужно было запускать другой проигрыватель. Разве все работает не так?»

«Ну да, это работает нормально, — ответила Рут, — есть ряд других проблем. Я могу запустить воспроизведение щелчком по кнопке запуска, но у меня нет способа сделать паузу или остановить воспроизведение. Я вынуждена слушать все 30 секунд фрагмента. Кроме того, все фрагменты содержат начало песни. У некоторых песен очень длинное вступление, поэтому по их началу нельзя получить представление о самой песне. Фрагмент должен начинаться где-то в середине этих песен, чтобы пользователи могли понять, что собой представляет песня. Также фрагмент воспроизводится на полной громкости, а затем резко прекращается. Если динамики пользователя настроены на высокий уровень громкости, воспроизведение может быть слишком громким. Я думаю, что будет лучше постепенно увеличивать громкость в начале и уменьшать в конце фрагмента».

Гуатам был немного разочарован. «Нужно было мне сказать все это раньше, когда мы это обсуждали. Ты дала не очень много информации, поэтому мне пришлось многое додумывать самостоятельно. Я могу все это сделать, но на это потребуется еще несколько дней».

Самое лучшее хранилище требований бесполезно, если оно не содержит высококачественной информации. Эта глава описывает необходимые характеристики требований и документов требований. Приведено много рекомендаций по написанию требований вместе с примерами неправильных требований и предложений по их исправлению. Эти рекомендации применимы

к требованиям, создаваемым в проектах независимо от применяемых в них методиках разработки. Авторы требований в каждом проекте должны самостоятельно определять необходимый уровень точности и детализации своих требований, но ничто не заменит ясной и четкой коммуникации.

Характеристики превосходных требований

Как отличить хорошие требования от проблемных? В этом разделе описываются несколько характеристик, которыми должны отличаться отдельные требования, после чего рассказывается о желательных характеристиках требований в целом (Davis, 2005; ISO/IEC/IEEE, 2011). Лучший способ сказать, обладают ли требования необходимыми атрибутами, — попросить нескольких заинтересованных лиц проверить их. Разные заинтересованные лица обнаружат разные недостатки. В главе 17 описывается использование контрольных списков для напоминания рецензентам о стандартных ошибках в требованиях.

Характеристики отдельных положений спецификации требований

В идеальном мире каждый отдельный пользователь, компания или функциональные требования отвечают различным параметрам качества, которые и описаны в следующих разделах.

Полнота

Каждое требование должно содержать всю информацию, необходимую читателю, чтобы понять его. В случае функциональных требований это означает предоставление всей информации, необходимой разработчику, чтобы реализовать их. Если вы понимаете, что данных определенного рода не хватает, используйте пометку «TBD» (to be determined — необходимо определить) на полях как стандартный флаг для выделения пробелов или зафиксируйте их в системе отслеживания дефектов, чтобы вернуться к ним позже. Восполните все пробелы в каждом фрагменте требований, прежде чем разработчики приступят к разработке этой функции.

Корректность

Каждое требование должно точно описывать возможность, которая будет удовлетворять какую-то потребность заинтересованного лица и четко определять функциональность, которую надо построить. Для соблюдения корректности необходима связь с источниками требований, в качестве которых могут выступать пользователь, предоставивший начальное требование, высокоуровневое системное требование, вариант использования, бизнес-правило или другой документ. Низкоуровневое требование, которое конфликтует со своим родительским требованием нельзя считать корректными. Однако

основная оценка здесь — за представителями пользователей или их непосредственным заместителям.

Осуществимость

Необходима возможность реализовать каждое требование при известных возможностях и ограничениях системы и рабочей среды, а также в рамках временных, бюджетных и ресурсных ограничений проекта. Разработчик, участвующий на этапе выявления требований может выполнить проверку на предмет того, что можно и что нельзя выполнить с технической точки зрения, а на что требуются только исключительные затраты и усилия. Инкрементальная разработка и прототипы проверки концепции — два способа проверить осуществимость требования. Если требование нужно удалить, потому что оно неосуществимо, нужно понимать его влияние на концепцию и границы проекта.

Необходимость

Каждое требование должно отражать возможность, которая действительно предоставит заинтересованным лицам ожидаемую бизнес-пользу, выделит продукт на рынке или необходима для соблюдения внешних стандартов, политик или правил. Каждое требование должно поступать из источника, которое имеет полномочия на предоставление требований. Отследите каждое требование вплоть до стадии сбора мнений пользователей, когда выявлялись варианты использования или пользовательские истории. Вы должны отследить каждое требование до бизнес-целей, которые четко определяют, почему оно необходимо. Если кто-то спросит, почему включено то или иное требование, у вас будет наготове обоснованный ответ.

Назначение приоритетов

Определяйте приоритеты бизнес-требований на основании важности для получения требуемой пользы. Назначьте приоритеты реализации каждому функциональному или пользовательскому требованию, потоку вариантов использования или функции, чтобы определить, насколько они важны для конкретного выпуска продукта. Если все требования одинаково важны, менеджеру проекта будет трудно справиться с нарушением сроков, потерей персонала или добавлением новых требований в процессе разработки. Приоритизация требований должна выполняться коллективно, чтобы учесть интересы многих заинтересованных лиц. Подробнее о назначении приоритетов см. главу 16.

Недвусмысленность

Естественный язык несет в себе два типа двусмысленности. Первый тип я могу определить самостоятельно, когда прикидываю, нет ли более одного способа интерпретации данного требования. Другой тип двузначности поймать сложнее. Это происходит, когда разные люди читают одно требование,

но интерпретируют его по-разному. Они находят требование содержательным, но оно имеет для них разное значение. Рецензирование — хороший способ обнаружить двусмысленности (Wiegers, 2002). Формальное рецензирование коллегами, такое как освидетельствование (в отличие от простой передачи требований коллегам для самостоятельного изучения), предоставляет возможность каждому участнику сравнить свое понимание каждого требования с другими. Понятие «исчерпывающее» связано с понятием «недвусмысленное»: читатели должны понимать, о чем идет речь в требовании. Процесс дружественного рецензирования описан в главе 17.

Вам никогда не удастся полностью устранить двусмысленность из требований — такова природа человеческого языка. В большинстве случаев разумные люди делают правильные выводы даже из немного туманных требований. Но помощь со стороны коллег в виде рецензирования позволит избавиться от многих самых плохих проблем.

Проверяемость

Сможет ли тестировщик разработать тесты или применить другие приемы, чтобы установить, действительно ли в продукте реализовано каждое требование? Если требование не поддается проверке, вопрос корректности его реализации становится предметом оценки, а не объективного анализа. Неполные, несогласованные, невыполнимые или двусмысленные требования также не поддаются проверке. Тестировщики знают толк в рецензировании требований на пригодность для проверки. Включите их в рецензенты своих требований, чтобы обнаружить проблемы как можно раньше.

Характеристики наборов требований

Недостаточно получить прекрасные отдельные формулировки требований. Набор требований, составляющий базовое соглашение о требованиях (baseline) для определенного выпуска или итерации, должен отвечать характеристикам, описанным в следующих разделах, будь они записаны в документе спецификаций требований к ПО, средстве управления требованиями, наборе пользовательских историй и приемочных тестах или любой другой форме.

Полнота

Никакие требования или необходимые данные не должны быть пропущены. На практике никогда не документируются *все и каждое* требование к системе. Всегда есть предполагаемые и подразумеваемые требования, хотя они и более рисковые, чем явно сформулированные требования. Недостающие требования сложно обнаружить, потому что их просто нет! В разделе «Предотвращение неполноты» далее в этой главе предлагаются некоторые способы определения отсутствующих требований. Любая спецификация, содержащая записи TBD является неполной.

Согласованность

Согласованные требования не конфликтуют с другими требованиями того же типа или с высокоуровневыми пользовательскими, системными или бизнес-требованиями. Если не устранить несогласованность требований до начала процесса разработки, ее придется устранять разработчикам. Рекомендуется записывать автора каждого требования, чтобы узнать, кто его высказал, если будет обнаружен конфликт. Выявить несоответствия может быть сложным, если соответствующая информация хранится в разных местах, например в документе концепции и границ проекта и в средстве управления требованиями.

Способность к модификации

Требование всегда можно переписать, но нужно поддерживать историю изменений каждого требования, особенно после достижения базового соглашения о требованиях. Нужно также знать о связях и зависимостях между требованиями, чтобы можно было найти те, что должны меняться одновременно. Возможность модификации диктует, что каждое требование должно уникально именоваться и выражаться отдельно от других, чтобы на него можно было однозначно ссылаться из других требований. Подробнее о разных способах именования требований см. главу 10.

Для сохранения возможности модификации избегайте излишней информации в требованиях. Повторяя требования в нескольких местах спецификации требований к ПО, где они уместны, вы облегчаете чтение документа, но затрудняете его поддержку. Одинаковые требования придется изменять одновременно, в противном случае возникнет несогласованность. Перекрестные ссылки на связанные между собой элементы в спецификации помогут вам синхронизировать последние при внесении изменений. Храня отдельные требования в средстве управления требованиями или в базе данных только один раз, вы решите проблему избыточности и упростите повторное использование требований, общих для нескольких проектов. Подробнее о повторном использовании требований см. главу 18.

Отслеживаемость

Отслеживаемое требование можно отследить как назад, к первоисточнику, так и вперед, к производным требованиям, элементам дизайна и исходному коду, который его реализует, а также к тестам, которые позволяют проверить корректность реализации. Заметьте, что на самом деле не обязательно определять все эти связи для требования, чтобы получить свойства, обеспечивающие возможность отслеживания требования. Отслеживаемые требования обозначаются постоянными уникальными идентификаторами. Они записаны в структурированной, детализированной форме, в противоположность длинным абзацам в повествовательной форме. Избегайте соединения нескольких требований в одну формулировку, потому что разные требования

могут быть связаны с различными элементам разработки. Подробнее об отслеживании связей требований см. главу 29.

Вам никогда не удастся создать идеальную спецификацию, в которой бы у *всех* требований присутствовали *все* эти атрибуты. Однако если вы будете помнить об этих характеристиках при написании и проверке требований, вы создадите более качественную спецификацию требований, а значит, и программный продукт.

Принципы создания требований

Не существует выверенного способа написания идеальных требований, а лучший учитель — это опыт, который нарабатывается со временем. Конструктивная критика проницательных коллег — огромное подспорье, потому что позволяет понять, когда ваш текст выполняет свою задачу, а когда нет. Вот почему рецензирование документов требований коллегами так важно. Начните процесс рецензирования с дружбы с коллегой бизнес-аналитиком и взаимного рецензирования требований. Вы узнаете, как другой бизнес-аналитик пишет требования, и внесете свой вклад в успех команды за счет более раннего обнаружения ошибок и возможностей улучшения. В следующих разделах приводится много советом по написанию требований — особенно функциональных — так, чтобы читатели их понимали. Benjamin Kovitz (1999), Ian Alexander и Richard Stevens (2002) и Karl Wiegers (2006) предоставляют многие другие рекомендации и примеры написания хороших требований.

Говоря «написание требований», люди обычно подразумевают текстовое представление требований на естественном языке. Но лучше мысленно перефразировать «написание требований» на «представление знания о требованиях». Во многих случаях альтернативные варианты представления информации доводят ее более эффективно, чем простой текст (Wiegers, 2006). Бизнес-аналитик должен выбрать подходящее сочетание методов коммуникации, которые гарантируют ясное и единое понимание как потребностей заинтересованного лица, так их решения, которое планируется создать.

Представленные здесь примеры требований всегда можно улучшить, и всегда есть эквивалентные способы их представления. При написании требований преследуются две важных цели:

- любой, прочитавший требование, должен придти к той же интерпретации, что и любой другой читатель этого требования;
- интерпретации всех читателей требования должны совпадать с тем, что автор хотел передать.

Эти результаты важнее, чем чистота стиля и догматическое следование каким-то заданными правилам и соглашениям.

Системная или пользовательская точка зрения

Функциональные требования можно писать с точки зрения того, что делает система или что делает пользователь. Так как важнее всего эффективно передать информацию, нет ничего страшного в смешении этих подходов и формулировании каждого требования в максимально понятной форме. Требования следует излагать последовательно, например «Система должна» или «Пользователь должен», затем — активный глагол, а после — наблюдаемый результат. Укажите инициирующие условия или триггеры, вследствие которых система ведет себя определенным образом. Общий шаблон требования, написанного с точки зрения системы, таков (Mavin et al., 2009):

[*необязательное предварительное условие*] [*необязательный триггер события*] *система должна [ожидаемая реакция системы*].

Шаблон взят из статьи «Easy Approach to Requirements Syntax (EARS)». EARS также содержит дополнительные конструкции шаблонов для нежелательного поведения, необязательных и сложных требований, а также требований, управляемых событиями или состояниями. Вот пример простого функционального требования, описывающего действие системы, на основе этого шаблона:

«Если запрошенный химикат есть на складе химикатов, система должна отобразить список всех хранимых на складе контейнеров с указанным химикатом».

Этот пример содержит предварительное условие, но в нем нет триггера. Некоторые авторы требований опускают из требований фразу «система должна», аргументируя это тем, что поскольку требования описывают поведение системы, нет необходимости постоянно упоминать, что «система должна». В данном примере удаление фразы «система должна» не изменит смысла. Иногда естественнее выразить требование в терминах действий пользователя, а не с точки зрения системы. Включение «должен» и использование активного залога позволяет четко сказать, какая сущность выполняет действие.

При написании функциональных требований с точки зрения пользователя хорошо работает следующая общая структура (Alexander и Stevens, 2002):

[*класс пользователя или имя действующего лица*] *должен иметь возможность* [*выполнить что-то*] [*с каким-то объектом*] [*условия выполнения, время отклика или декларация качества*].

Альтернативная формулировка выглядит так: «Система должна позволять (или разрешать или давать возможность) [название класса пользователя] [делать что-то].» Вот пример функционального требования, написанного с точки зрения пользователя:

Химик должен иметь возможность повторно заказать любой химикат, который он ранее заказывал, путем извлечения и редактирования параметров ранее введенного заказа.

Заметьте, что в требовании используется название класса пользователя — Химик, а не общий термин «пользователь». Явная формулировка максимально снижает вероятность неверного истолкования.

Язык и стиль

Написание требований не похоже на создание художественных и нехудожественных произведений. В данном случае не подходит стиль письма, к которому вы привыкли в школе и институте и в котором сначала излагается основная идея, потом описываются подкрепляющие основную идею факты и, наконец, формулируется заключение. Надо изменить стиль письма: первой формулируется суть — формулировка потребности или функциональности, после чего следуют детали (обоснование, источник, приоритет и другие атрибуты требования). Такая структура удобная как читателям, которые бегло просматривают документ, так и тем, кому нужны все детали. Включение таблиц, структурированных списков, диаграмм и других визуальных элементов помогает оживить монотонное перечисление функциональных требований и предоставляет более богатую информацию тем, кому проще осваивать материал нетрадиционными методами.

Документы требований также не являются местом, где можно упражняться в креативности и изобретательности в плане стиля. Избегайте смешения активного и пассивного залогов в попытке сделать материал более интересным для чтения. Не обозначайте одно понятие разными терминами, чтобы разнообразить свой текст (клиент, покупатель, посетитель, пользователь, заказчик). Простота чтения и понимания — жизненно важный элемент качественно написанного требования, а вот интересность и увлекательность, честно говоря, не так важны. Если у вас нет опыта, следует ожидать, что ваши читатели не всегда смогут понять, что вы хотели до них донести. Следуйте приведенным ниже советам при работе над формулировками своих требований, чтобы они максимально эффективно передавали информацию целевой аудитории.

Ясность и лаконичность Пишите требования полными предложениями, с правильной грамматикой, правописанием и пунктуацией Предложения и абзацы должны быть краткими и ясными. Пишите требования простым и прямолинейным языком, характерным для соответствующей предметной области, но не используйте профессиональный жаргон. Определения используемых терминов размещайте в словаре терминов.

Еще одна хорошая привычка — писать кратко. Фразы типа «должна предоставлять пользователю возможность сделать» можно свести к одному слову — «должна». Для каждой включаемой в требования порции информации задавайте себе вопрос: «Что читатель будет делать с этой информацией?» Если вы не уверены, что ваша информация будет полезной каким-либо заинтересованным лицам, скорее всего она не нужна. Вместе с тем ясность важнее лаконичности.

Точно сформулированные требования повышают вероятность того, что люди получат то, что ожидали; менее точные требования оставляют разработчику больше пространства для свободы интерпретации. Иногда подходит и не очень большая точность, но в других случаях это может приводить к

слишком большому отклонению от желаемого результата. Если разработчик, рецензирующий спецификацию требований к ПО, не совсем понимает цели заказчика, следует включить дополнительную информацию, чтобы снизить риск возникновения проблем в дальнейшем.

Ключевое слово «должна» Традиционно при описании какой-либо возможности системы используют слово «должна». Некоторым это не нравится. Ну и что? Формулировки со словом «должна» четко говорят о требуемой функциональности — в полном соответствии с самой важной целью ясного и четкого доведения информации. Вам может больше понравиться другие слова, такие как «обязана», «следует» или что-то похожее, но нужно придерживаться одного термина. Иногда мне приходится читать спецификации, содержащие произвольные и вводящие в заблуждения наборы глаголов: должна, может, будет, надлежит, должна бы, могла бы, обязана, надо бы, должна бы обеспечить и т. п. Никогда не знаешь, есть различия в значении этих глаголов или нет. Нюансы смысла различных глаголов также сильно усложняют единообразную интерпретацию документа в командах, члены которых относятся к разным культурам. Лучше всего придерживаться ключевого слова «должна» (shall).

Некоторые авторы требований специально пользуются разными глаголами для обозначения небольших смысловых различий. Для обозначения приоритета они используют определенные ключевые слова: «должна» означает обязательность, «будет» означает желательную, а «может» — необязательную функциональность (ISO/IEC/IEEE 2011). Мы считаем такие условности опасными. Намного понятнее, когда четко формулируется — «должна» и явно задается приоритет требования: высокий, средний или низкий. Также приоритеты могу меняться по мере завершения итераций, поэтому не нужно их указывать в формулировке требований. Сегодняшнее «должна» завтра может стать «надо бы». Другие авторы используют «должна» в требованиях, а для обозначения ожидаемого поведения дизайна применяют «будет». При использовании таких соглашений есть риск, что некоторые читатели не поймут разницы между словами, которые в повседневном общении применяются практически в одном значении, поэтому такой разноголосицы нужно избегать.

> **Внимание!** Один остроумный консультант предложил мысленно заменить «может» на «скорее всего не» и посмотреть на такое видоизмененное требование. Если такая формулировка неверна, то «может» надо заменить на более точное слово.

Активный залог Используйте активный залог, чтобы дать точно понять, какая сущность выполняет описываемое действие. Много научных и бизнес-текстов написаны с использованием пассивного залога, но он никогда не бывает таким ясным и прямым, как активный залог. Следующее требование написано с использованием пассивного залога:

После доставки обновления продукта будет изменен регистрационный номер в соответствующей строке договора.

Фраза «будет изменен» — признак пассивного залога. Она обозначает адресата (регистрационный номер), но не исполнителя действия. То есть эта фраза ничего не говорит о том, кто или что обновляет регистрационный номер. Выполняет ли это система автоматически или регистрационный номер должен обновить пользователь? Изменение формулировки этого требования с использованием активного залога явно показывает действующее лицо и уточняет инициирующее событие:

Когда отдел обслуживания подтвердит, что они отправили обновление продукта, система должна обновить клиентский договор, задав в нем новый регистрационный номер продукта.

Раздельные требования Избегайте длинных повествовательных абзацев, которые содержат несколько требований. Читатели не должны вылавливать изюминки отдельных требований из единой безликой массы текста. Четко отделяйте требования от вводной и контекстной информации. Такая информация полезна читателям, но они должны четко отличать формулировки требований. Как-то мне пришлось рецензировать объемную спецификацию требований, написанную в форме длинных абзацев. Я читал страницы «цельнолитого» текста и понимал его, но было очень сложно выловить в нем конкретные требования. Другие читатели могут придти к неправильным выводам относительно того, где в этом тексте скрываются зерна требований.

Наличие в требовании таких слов, как «и», «или», «дополнительно» и «также», предполагает, что несколько требований могли быть объединены. Это не означает, что нельзя использовать союз «и», но если вы делаете это, проверяйте, соединяет ли он две части одного требования или два отдельных требования. Если для проверки двух частей требуются разные тесты, разбейте предложение на отдельные требования.

Никогда не используйте «и/или» в требованиях; это оставляет читателю свободу интерпретации, вот пример:

Систем должна поддерживать поиск по номеру заказа, номеру счета и/или по номеру пользовательского заказа на покупку.

Такое требование позволило бы пользователю ввести одно, два или три числа при выполнении одной операции поиска. Скорее всего, это не то, что планировалось.

Такие выражения, как «пока не» и «кроме» также указывают на наличие нескольких требований:

Кредитная карточка покупателя должна считаться действительной для платежей до тех пор, пока не истечет ее срок действия.

Отсутствие информации о том, что происходит, когда выполняется условие «пока не», — обычная причина наличия недостающих требований. Разделите это положение на два — для двух условий: когда кредитная карточка действительна и когда срок ее действия истек:

Если кредитная карточка покупателя действительна, система должна выполнить платеж по этой карточке.

и

Если срок действия кредитной карточки покупателя истек, система должна предоставить покупателю возможность обновить информацию своей кредитной карты или ввести для платежа информацию другой кредитной карточки.

Уровень детализации

Требования должны определяться настолько подробно, чтобы предоставлять разработчикам и тестировщикам ровно столько информации, сколько необходимо для их реализации.

Достаточный уровень детализации Важной частью анализа требований является разбиение высокоуровневого требования на части, столь мелкие, чтобы они объясняли и дополняли суть происходящего. Нет универсального ответа на обычно задаваемый вопрос: «Насколько детальными должны быть требования?» Требования должны быть сформулированы достаточно подробно, чтобы риск непонимания был минимальным, для этого необходимо учесть знания и опыт разработчиков. Чем меньше возможностей для обсуждения проблем с текущими требованиями, тем больше деталей нужно включить в набор требований. Если разработчики предлагают несколько способов удовлетворения требования и все они приемлемы, значит, особенности продукта и детализация изложения выбраны верно. Дополнительные детали требуются, когда (Wiegers, 2006):

- работа выполняется для внешнего клиента;
- разработка или тестирование будет передаваться на сторону;
- члены проектной команды распределены географически;
- тестирование системы будет выполняться на основе требований;
- необходимы точные оценки;
- требуется отслеживать связи требований.

Можно сократить объем деталей, когда:

- работа выполняется для внутренних нужд компании;
- клиенты активно участвуют в разработке;
- у разработчиков значительный опыт и знания предметной области;
- есть прецеденты и когда новое приложение создается на замену предыдущему;
- будет использоваться пакетное решение.

Единый уровень детализации Создатели документации зачастую тратят массу сил, чтоб «поймать» нужный уровень детализации. Не обязательно описывать все ваши требования с одним уровнем детализации. В частности, можно подробнее описывать те области, которые характеризуются повышенным риском. Тем не менее, в наборе связанных требований лучше придерживаться одного уровня детализации при описании функциональных требований.

Попробуйте описать требования, которые можно протестировать по отдельности. Число поддающихся тестированию требований были предложены в качестве метрики, определяющей размер ПО (Wilson, 1995). Если вам удастся придумать небольшое число взаимосвязанных вариантов тестирования, чтобы проверить правильность реализации требования, скорее всего необходимый уровень детализации достигнут. Если ваши тесты многочисленны и разнообразны, вероятно, несколько требований соединены вместе. Их следует разделить на более простые.

Мне приходилось видеть в одной и той же спецификации положения, которые значительно варьировались в их границах. Например, следующие две функции были разбиты на отдельные требования:

1. *Комбинация клавишей Ctrl+S должна интерпретироваться как «Сохранить файл».*
2. *Комбинация клавишей Ctrl+P должна интерпретироваться как «Печать файла».*

Это очень детальные требования. Для проверки правильного поведения потребуется очень мало тестов. Представьте себе несуразно длинный список подобных требований, который лучше выразить в виде таблицы, перечисляющей сочетания клавиш и соответствующие действия системы.

Вместе с тем та же спецификация требований содержала крупное функциональное требование:

Продукт должен реагировать на команды редактирования, введенные голосом.

Это простое требование — по виду не больше и не меньше остальных мелких требований в спецификации требований — предусматривает включение сложной подсистемы распознавания речи — практически целый самостоятельный продукт! Проверка одного этого требования в работающей системе потребует сотен тестов. Такая формулировка требования уместна на высоком уровне абстракции, который уместен в документе концепции или маркетинговых требований, но требование по распознаванию речи совершенно точно требует больше подробностей функциональности.

Способы представления

Глаза читателя не цепляются за плотную массу объемного текста или длинного списка единообразных требований. Подумайте над тем, как самым эффективным образом донести каждое требование до соответствующей целевой аудитории. Альтернативами выражения требований на естественном языке являются списки, таблицы, наглядные модели анализа, графики, математические формулы, фотографии, звуковые и видеоклипы. Во многих случаях альтернативы не избавляют от необходимости текстового представления требований, но они служат превосходной вспомогательной наглядной информацией, способствующей более глубокому пониманию предмета читателем.

Как-то я проверял требования, которые были представлены в следующем формате:

Текстовый редактор должен уметь анализировать документы следующего <формат>, определяющих законы <юрисдикции>.

Предлагалось три возможных значения *<формат>* и четыре возможных значения *<юрисдикции>* для двенадцати схожих требований. Спецификация требований к ПО действительно содержала 12 требований, однако из этих двенадцати требований одного не хватало, а одно повторялось. Найти ошибку можно было, составив таблицу всех возможных вариантов, которая компактнее и менее однообразная, чем список требований. Общее требование можно было сформулировать так:

Editor.DocFormat *Текстовый редактор должен уметь анализировать документы нескольких форматов, определяющие законы юрисдикций, как показано в табл. 11-1.*

Табл. 11-1. Требования по анализу документов

Юрисдикция	Теговый формат	Безтеговый формат	Формат ASCII
Федеральная	.1	.2	.3
Региональная (на уровне штата)	.4	.5	.6
Территориальная (местный)	.7	Неприменимо	.8
Международная	.9	.10	.11

Ячейки таблицы содержат только суффикс, который присоединяется к идентификатору основного требования. Например, третье требование в верхней строке преобразуется в следующее:

Editor.DocFormat.3 *Текстовый редактор должен уметь анализировать документы ASCII, определяющие федеральные законы.*

Если по определенным причинам у каких-то сочетаний нет соответствующих функциональных требований, укажите в соответствующих ячейках таблицы «Неприменимо». Это намного понятнее, чем пропустить соответствующее сочетание в списке и заставить читателя гадать, почему нет требования по анализу документов местных законов в безтеговом формате. Этот прием также гарантирует полноту набора требований — если нет пустых ячеек, можно быть уверенным, что вы ничего не пропустили.

Предотвращение неопределенности

Качество требований определяет читатель, а не автор. Бизнес-аналитик может считать, что написанное им требование кристально-ясное, свободно от неоднозначностей и других проблем. Но если у читателя возникают вопросы, требование нуждается в дополнительном совершенствовании. Рецензирование — лучший способ поиска мест, где требования непонятны всем целевым аудиториям. В этом разделе рассказывается о популярных причинах неоднозначности требований.

Неоднозначные слова Последовательно используйте термины и именно так, как они определены в словаре. Остерегайтесь синонимов и слов, близких по значению. Я видел проект, в котором для обозначения одного элемента в одном и том же документе требований использовались четыре разных термина. Выберите один термин и последовательно используйте только его, а синонимы укажите в словаре терминов, чтобы люди, привыкшие к другому названию, могли связать его с вашим термином.

При использовании местоимения, ссылающегося на что-то упомянутое ранее, обеспечьте, чтобы было четко понятно, на что вы ссылаетесь. Наречия привносят субъективность, а, значит, неоднозначность. Избегайте таких слов, как *разумно*, *уместно*, *вообще*, *приблизительно*, *обычно*, *систематично* и *быстро*, потому что нет уверенности, что читатель сынтерпретирует их правильно.

Требования, изложенные неясным языком, не поддаются проверке, поэтому избегайте двусмысленных и субъективных терминов. В табл. 11-2 перечислены многие из них, а также приводятся рекомендации, как исправить такие неясности. Некоторые из этих слов могут применяться в бизнес-требованиях, но только не в пользовательских или конкретных функциональных требованиях, описывающих решение, которое планируется создать.

Табл. 11-2. Некоторые неоднозначные термины, которых следует избегать в спецификации к требованиям

Неоднозначные термины	Способы улучшения
Приемлемый, адекватный	Определите, что понимается под приемлемостью и как система это может оценить
И/или	Укажите точно, что имеется в виду — «и» или «или», чтобы не заставлять читателя гадать
Практически выполнимо	Не заставляйте разработчиков определять, что под этим понимается. Поставьте пометку «TBD» и определите дату, к которой эту проблему следует разрешить
По меньшей мере, как минимум, не более чем, не должно превышать	Укажите минимальное и максимальное допустимые значения
Наилучший, самый больший, большинство	Укажите, какой уровень требуется, а также минимальный приемлемый уровень
Между, от X до Y	Укажите, входят ли конечные точки в диапазон
Зависит от	Определите природу зависимости. Обеспечивает ли другая система ввод данных в вашу систему, надо ли установить другое ПО до запуска вашей системы и зависит ли ваша система от другой при выполнении определенных расчетов или служб?

Табл. 11-2. (*продолжение*)

Неоднозначные термины	Способы улучшения
Эффективный	Определите, насколько эффективно система использует ресурсы, насколько быстро она выполняет определенные операции и как быстро пользователи с ее помощью могут выполнять определенные задачи
Быстрый, скорый, моментальный	Укажите минимальное приемлемое время, за которое система выполняет определенное действие
Гибкий, универсальный	Опишите способы адаптации системы в ответ на изменения условий работы, платформ или бизнес-потребностей
Улучшенный, лучший, более быстрый, превосходный, более качественный	Определите количественно, насколько лучше или быстрее должны стать показатели в определенной функциональной области или аспект качества
Включает; включает в себя, но не ограничен этим; и т. д.; и т. п.; такой как, в частности	Перечислите все возможные значения или функции, а не только примеры, или укажите читателю, где можно найти исчерпывающий список. В противном случае разные читатели могут по-разному понимать, что должен содержать полный список и где он должен заканчиваться
В большинстве случаев, обычно, как правило, практически всегда	Уточните, когда указанные условия или сценарии неприменимы и что должно происходить в таком случае. Опишите, как пользователь или система должны различать разные случаи
Соответствует, равняется, согласуется, представляет то же самое	Определите, должно ли сравнение текста быть чувствительным к регистру и что означает эта фраза — «содержит», «начинается с» или «точно совпадает». Для действительных чисел определите точность при сравнении
Максимизируйте, минимизируйте, оптимизируйте	Укажите минимальное и максимальное допустимые значения определенного параметра
Обычно, в идеальном варианте	Опишите нештатные или неидеальные условия и как система должна вести себя в таких ситуациях
Необязательно	Укажите, кто делает выбор: система, пользователь или разработчик
Возможно, желательно, должно	Должно или не должно?
Разумный, при необходимости, когда уместно, по возможности	Объясните четко, как разработчик или пользователь должен оценивать разумность и уместность

Табл. 11-2. (*окончание*)

Неоднозначные термины	Способы улучшения
Устойчивый к сбоям	Определите, как система должны обрабатывать исключения и реагировать на неожиданные условия работы
Цельный, прозрачный, корректный	Что означает «цельный» или «корректный» для пользователя? Выразите ожидания пользователя, применяя характеристики продукта, которые можно наблюдать
Несколько, некоторые, много, немного, множественный	Укажите сколько или задайте минимальную и максимальную границы диапазона
Не следует	Старайтесь формулировать требования в позитивной форме, описывая, что именно система будет делать
Современный	Поясните этот термин для заинтересованного лица
Достаточный	Укажите, какая степень чего-либо свидетельствует о достаточности
Поддерживает, позволяет	Дайте точное определение, из каких действий системы состоит «выполнение» конкретной возможности
Дружественный, простой, легкий	Опишите системные характеристики, которые будут отвечать потребностям пользователей и его ожиданиям, касающимся легкости и простоты использования продукта

Конструкция типа «A/B» Многие спецификации требований содержат выражения вида «A/B», то есть двух связанных терминов (синонимов или антонимов) разделенных косой чертой. Такие выражения как правило неоднозначны. Вот пример:

Система должна обеспечивать автоматический сбор информации о лицензионных ключах при массовом выпуске продукта отделом доставки/исполнения.

Это предложение можно истолковать несколькими способами:

- «отдел доставки/исполнения» — это название подразделения;
- доставка и исполнение являются синонимами;
- в некоторых проектах упомянутое подразделение называется отделом доставки, а в других — отделом исполнения;
- массовый выпуск продукта может выполнять отдел доставки или отдел исполнения, так что косая черта означает «или»;
- массовый выпуск продукта отдел доставки или отдел исполнения выполняют совместно, так что косая черта означает «и».

Иногда авторы применяют конструкцию «A/B», потому что не совсем уверены, что им хочется сказать. К сожалению, это означает, что каждый читатель делает собственные выводы, в чем собственно заключается требование. Лучше точно решить, что хочется сказать, и выразить это правильными словами.

Пограничные значения Много неоднозначности возникает на границах числовых диапазонов как в требованиях, так и бизнес-правилах. Посмотрите на следующее:

Отпуск длительностью до 5 дней не требует одобрения. Запросы на отпуск длительностью от 5 до 10 дней требуют одобрения непосредственного начальника. Отпуска длительностью 10 дней и более требуют одобрения директора.

При такой формулировке непонятно, в какую категорию попадают отпуска длительностью точно 5 и 10 дней. Если ввести дробные числа, скажем 5,5 дней отпуска, то дело еще больше запутается. Слова «от и до», «включительно» и «свыше» вносят четкость и ясность насчет пограничных значений:

Отпуск длительностью 5 дней и меньше не требует одобрения. Запросы на отпуск длительностью более 5 и до 10 дней включительно требуют одобрения непосредственного начальника. Отпуска длительностью свыше 10 дней требуют одобрения директора.

Негативные требования Иногда люди пишут в требованиях не то, система должна, а то, что она *не должна* делать. Как реализовать такие «негативные» требования? Особенно сложны в расшифровке двойные и тройные отрицания. Попытайтесь переформулировать негативные требования в позитивном стиле, которые описывает ограничение на поведение. Вот пример:

Пользователь не должен иметь возможность активизировать договор, если он не сбалансирован.

Лучше перефразировать это двойное отрицание («не должен» и «не сбалансирован») как позитивное утверждение:

Система должна позволять пользователю активировать договор, только если этот договор сбалансирован.

Вместо применения негативных требований для указания того, что определенная функциональность выходит за рамки проекта, включите это ограничение в раздел «Ограничения и исключения» документа концепции и границ проекта, как описано в главе 5. Если какое-то требование было в границах проекта, но потом было удалено, лучше не терять его из вида — оно может потребоваться когда-нибудь в будущем. Если требования размещаются в документе, используйте зачеркивание, чтобы отметить удаленное требование. Лучше всего при работе с такими удаленными требованиями использовать атрибут состояния в системе управления требованиями (подробнее об отслеживании атрибутов и состояния требований см. главу 27).

Предотвращение неполноты

Нет никакого способа узнать наверняка, что обнаружены все и каждое требование. В главе 7 описывается несколько способов определения отсутствующих требований. Ориентация на выявление пользовательских задач, а не функций системы помогает не упустить функциональность. Использование моделей анализа также помогает обнаружить пропущенные требования (см. главу 12).

Симметрия Симметричные операции — типичный источник пропущенных требований. Как-то я обнаружил в рецензируемой спецификации требований к ПО следующее требование:

Пользователь должен иметь возможность сохранить договор в любой момент ручного ввода договора.

Мне не удалось найти в спецификации требование, предусматривающее для пользователя возможность открыть ранее сохраненный неполный договор для завершения работы над ним — скорее всего это требование упустили. Также ничего не говорилось о том, должна ли система перед сохранением проверять данные, введенные в договор. Может это неявное требование? Разработчикам нужно точно знать ответ на этот вопрос.

Сложная логика Сложные логические выражения часто оставляют неопределенными некоторые значения, необходимые для принятия решения. Посмотрите на следующее требование:

Если не выбран план «Премиальный» и не предоставлено подтверждение страховки, клиент должен автоматически получать план «Базовый».

Это требование ссылается на два двоичных выбора, которые дают четыре возможных сочетания. Но спецификация описывает только одно сочетание. В ней не говорится, что нужно делать, если:

- выбран план «Премиальный», а подтверждение страховки не предоставлено;
- выбран план «Премиальный» выбран и предоставлено подтверждение страховки;
- не выбран план «Премиальный», а подтверждение страховки предоставлено.

Читатель вынужден думать, что в таких ситуациях система не должна предпринимать никаких действий. Это может быть верно, но лучше это сформулировать явно, а не подразумевать. Для представления сложной логики используйте таблицы и деревья принятия решений, чтобы быть уверенным, что не пропустили какие-то варианты.

Отсутствующие исключения У каждого требования, описывающего, как система должна работать, когда все хорошо, должно быть сопутствующее требование, рассказывающее, как поступать в случае исключения. Посмотрите на следующее:

Если пользователь работает над существующим файлом и решит сохранить его, система должна сохранить файл под тем же именем.

Само по себе это требование ничего не говорит о том, что система должна делать, если не может сохранить файл под тем же именем. Соответствующее второе требование, идущее в паре с первым, может формулироваться так:

Если система не может сохранить файл под определенным именем, она должна предоставить пользователю возможность сохранить файл под другим именем или отменить операцию сохранения.

Примеры требований: до и после

В начале главы перечислено несколько характеристик качественно сформулированных требований. Поскольку наличие требований, не отвечающим этим характеристикам, приводит к неразберихе, напрасно затраченным усилиям и последующим переделкам, старайтесь разрешить проблемы на ранних стадиях работы. Далее я покажу несколько, далеких от совершенства требований, взятых из реальных проектов. Исследуйте каждое из них, используя перечисленные ранее характеристики качества, и попытайтесь определить проблемы. Для начала, например, выясните, поддаются ли они проверке. Если вам не удастся составить тесты, чтобы точно сказать, были ли требования реализованы соответствующим образом, возможно, они неясно сформулированы или не хватает необходимой информации.

Мы высказали свои соображения о проблемах для каждого требования и предложили несколько путей их решения. Дополнительные проверки еще их улучшат, однако на определенном этапе вы должны приступить к непосредственному написанию ПО. Больше примеров по исправлению неудачных требований изложено Ivy Hooks и Kristin Farry (2001), Al Florence (2002), Ian Alexander и Richard Stevens (2002), а также Karl Wiegers (2006). Заметьте, что извлечение требований из подобного контекста показывает их самые неприглядные стороны. Возможно, в исходной среде эти требования выглядели лучше. Мы также предполагаем, что бизнес-аналитики (и другие члены команды) приходят каждый день на работу, чтобы выполнить свою работу наилучшим образом в меру своего опыта и знание, так что наша цель здесь не в придирках к авторам.

> **Внимание!** Старайтесь избегать паралича аналитического процесса. Все примеры требований в состоянии «после» можно дополнительно улучшать, но нельзя тратить слишком много времени на попытки сделать требования идеальными. Помните, что ваша цель — написать требования, которые *достаточно хороши*, чтобы команд могла приступить к конструированию продукта при приемлемом уровне риска.

Пример 1. *Диспетчер фоновых задач должен предоставлять сообщения о состоянии через регулярные интервалы, составляющие не менее 60 секунд.*

Что понимается под сообщениями о состоянии? При каких условиях и как именно они поставляются пользователю? Как долго они должны отображаться на экране? Достаточно ли отображения на протяжении половины секунды? Продолжительность временного интервала не сформулирована точно, а слово «каждый» только вносит дополнительную неясность. Один из способов оценить требование — проверить, устраивают ли пользователя нелепые, но имеющие право на существование интерпретации этого требования. Если нет, то над требованием необходимо еще поработать. В этом примере интервал должен равняться *не менее* 60 секунд; таким образом, если сообщение будет появляться раз в год, это нормально? А если промежуток *не должен превышать* 60 секунд, то не будет ли интервал, составляющий одну миллисекунду, слишком коротким? Эти утрированные интерпретации

не выходят за рамки первоначального требования, но совершенно очевидно, что пользователь хотел совершенно другого. По этим причинам требование нельзя проверить.

А вот возможные способы исправления недостатков этого требования, которыми вы можете воспользоваться после того, как получите дополнительную информацию от клиента.

1. Диспетчер фоновых задач (ДФЗ) должен отображать сообщения о состоянии в определенной области пользовательского интерфейса.

1.1. ДФЗ должен обновлять сообщения каждые 60 плюс/минус 5 секунд после запуска фоновой задачи.

1.2. Сообщения должны оставаться видимыми все время, пока работает фоновая задача.

1.3. Если взаимодействие с фоновой задачей возможно, ДФЗ должен отображать процент выполнения фоновой задачи.

1.4. По завершению фоновой задачи ДФЗ должен отобразить сообщение «Выполнено».

1.5. Если фоновая задача «зависла», ДФЗ должен отобразить соответствующее сообщение.

При переписывании несовершенного требования его формулировка становится длиннее из-за необходимости включения отсутствовавшей информации. Разделение требования на несколько дочерних разумно, потому что для каждого понадобится отдельный тест, кроме того, так каждое проще отслеживать. Скорее всего будут дополнительные сообщения о состоянии, отображаемые ДФЗ. Если они задокументированы в другом месте, например в спецификации интерфейса, добавьте эту информацию по ссылке, не дублируя ее. Перечисление сообщений в таблице условий и соответствующих сообщений позволит предоставить информацию более лаконично, чем при написании множественных функциональных требований.

В измененном требовании не указан способ отображения сообщения о состоянии — указано просто «в определенной области пользовательского интерфейса». Такая формулировка делает размещение сообщений задачей дизайна, что допустимо в большинстве ситуаций. Если вы сейчас определите место отображения сообщений здесь, разработчики воспримут ее как ограничение. Излишние ограничения на дизайн расстраивают разработчиков, кроме того, в этом случае вряд ли можно рассчитывать на оптимальный продукт.

Но представьте, что мы добавляем эту функциональность в существующее приложение, у которого уже есть строка состояния, где пользователи привыкли видеть важные сообщения. Для единообразия имеет большой смысл предусмотреть условие, что сообщения о состоянии ДФЗ должны отображаться в строке состояния. То есть вы можете специально наложить ограничение дизайна, имея на то очень серьезную причину.

Пример 2. *Если возможно, номера счетов следовало бы проверять по списку корпоративных счетов.*

Что означает «если возможно»? Значит ли это «технически осуществимо» (вопрос для разработчика) или «когда доступен основной список счетов»? Если вы не уверены, можно ли реализовать функцию, сделайте пометку «TBD», чтобы указать, что эта проблема еще не решена. После проверки одно из двух будет ликвидировано — либо пометка «TBD», либо требование. В требовании не указано, что произойдет, если проверка пройдет успешно или окончится неудачей. Избегайте неточных слов вроде «следовало бы». Вот как выглядит исправленный вариант требования.

«В момент ввода номера счета система должна отобразить сообщение об ошибке, если этого номера счета в основном корпоративном списке счетов».

Связанное требование будет относиться к условию исключения: основной список счетов не доступен во время проверки.

Пример 3. *Тестер должен позволять пользователю легко подключать дополнительные компоненты, в том числе импульсный генератор, вольтметр, измеритель емкости и нестандартные тестовые платы.*

Это требования к продукту, содержащему встроенное ПО, которое используется для тестирования различных типов измерительных приборов. Слово «легко» подразумевает требование легкости и простоты использования, но оно не поддается ни измерению, ни проверке. «В том числе» не дает ясности, полный ли это список внешних устройств, которые должны подключаться к испытательному устройству, или существует еще множество других приборов, о которых мы не знаем. Вот какие альтернативные требования содержат некоторые преднамеренные ограничения дизайна.

1. *Тестер должен содержать USB-порт, чтобы пользователь смог подключить любое измерительный прибор, у которого есть USB-разъем.*
2. *USB-порт должен быть установлен на передней панели для того, чтобы позволить квалифицированному оператору подключить измерительный прибор за 10 секунд или менее.*

Бизнес-аналитик не должен по собственной инициативе переписывать требования так, чтобы они налагали ограничения дизайна. Вместо этого нужно стремиться обнаруживать несовершенные требования и обсуждать их с соответствующими заинтересованными лицами для уточнения требований.

Пример 4. *Система должна проверять наличие несоответствий данных счетов между журналом активных счетов и архивом диспетчера счетов. Логика, применяемая для создания этих сравнений, должна основываться на логике существующего средства проверки соответствия. Иначе говоря, код не нужно создавать с нуля. В качестве основы разработчики должны использовать код существующего средства проверки соответствия. Однако нужно добавить дополнительную логику определения базы данных, являющейся полномочным источником. Новая функциональность будет включать запись данных в таблицы, указывающие как и где разрешать несоответствия. Кроме того код должен проверять наличие исключительных сценариев по базе данных средств безопасности. При обнаружении несоответствий в команду безопас-*

ности должны отправляться автоматизированные уведомления о сообщениях электронной почты.

Это хороший пример для практики. Мы укажем на некоторые проблемы в этом разделе, а вы можете попытаться переписать его в поле правильной форме, внося свои предложения по заполнению пробелов. Вот некоторые проблемы, которые вы можете устранить.

- Есть ряд требований, который нужно разбить на отдельные требования.
- Если логика сравнения «основывается» на логике существующего средства проверки соответствия, то какая конкретно часть может повторно использоваться и как ее нужно изменить? Какие функции различаются в новой системе и в существующем средстве? Какую «дополнительную логику» нужно добавить? Как конкретно система может определить, какая база данных является «полномочным источником»?
- Новая функциональность «включает» запись данных в таблицы — это все или «включена» другая функциональность, о которой не сказано явно?
- Уточните, что означает «как и где», когда речь идет о разрешении несоответствий.
- В некоторых местах нужно использовать слово «должен».
- Какое соотношение между «исключительным сценарием» и «несоответствием»? Если это синонимы, выберите один термин и придерживайтесь именно его. В словаре терминов можно уточнить что это: одно и то же или разное и какая между ними связь.
- Какую информацию система должна отправлять в команду безопасности при обнаружении несоответствия?

Как уже говорилось ранее, вы никогда не сможете создать идеальные требования. Но опытный бизнес-аналитик может практически всегда помочь сделать требования лучше.

Что дальше?

- Проведите обсуждение со своими клиентами, разработчиками и тестировщикам, цель которого — оценить текущий уровень проектной документации по требованиям на предмет того, не нужно ли добавить или убрать часть подробностей в тех или иных областях и как представить эти требования.
- Проверьте страницу функциональных требований в спецификации требований к ПО вашего проекта и убедитесь, что каждое положение демонстрирует характеристики отличных требований. Проверьте, нет ли в требованиях проблем, описанных в этой главе. Перепишите любые требования, которые не отвечают этому уровню.
- Пригласите от трех до шести заинтересованных в проекте лиц для проверки спецификации требований к ПО для вашего проекта (Wiegers, 2002). Убедитесь, что каждое требование отвечает соответствующим

характеристикам, о которых говорилось в этой главе. Проверьте спецификацию на наличие конфликтов между различными требованиями в спецификации, недостающих требований и недостающих разделов. Убедитесь, что обнаруженные недостатки исправлены в спецификации требований к ПО и в любых предыдущих продуктах, созданных на основе этих требований.

Глава 12

Лучше один раз увидеть, чем 1024 раза услышать

Специалисты, работающие над проектом Chemical Tracking System, выполняли первую проверку спецификации требований к ПО. В ней участвовали Дейв (менеджер проекта), Лори (бизнес-аналитик), Хелен (ведущий разработчик), Рамеш (ведущий тестировщик), Тим (сторонник продукта от химиков) и Роксана (сторонник продукта от специалистов склада химикатов). Тим начал со слов: «Я прочитал всю спецификацию требований. Большинство требований мне показались нормальными, но в некоторых разделах было сложно переварить длинные списки требований. Я не уверен, что мы определили все этапы процесса заказа химиката».

«Мне было трудно продумать все варианты тестирования, которые необходимы, чтобы раскрыть все возможные изменения статуса заказа, — добавил Рамеш. — Я обнаружил, что ряд требований, касающихся изменения статуса, разбросаны по всей спецификации, но не могу твердо сказать, пропущены ли какие-то из них. Мне показалось, что пара требований противоречат друг другу».

Роксана обнаружила похожую проблему: «Меня сбило с толку описание того, как непосредственно выполнять заказ химиката, — сказала она. — Мне трудно представить всю последовательность действий при заказе».

После того как остальные участники поделились своими опасениями, Лори подвела итоги: «Похоже, спецификация не дает нам всего, что мы должны знать о системе. Я нарисую несколько диаграмм, чтобы нам было легче представить требования и понять, не прояснит ли это проблемные области. Спасибо за то, что высказали свое мнение».

Согласно авторитетному мнению специалиста в области требований Алана Девиса, никакое представление требований в одном виде не дает их полной картины (Davis, 1995). Необходима комбинация текстовых и визуальных способов представления требований на различных уровнях абстракции, чтобы получилась полная картина создаваемой системы. К таким способам относятся списки функциональных требований, таблицы, графические модели анализа, прототипы пользовательского интерфейса, приемочные тесты, де-

ревья и таблицы решений, видеоклипы и математические формулы (Wiegers, 2006). В идеале различные представления требований должны создавать разные специалисты. Бизнес-аналитик может написать функциональные требования и нарисовать часть моделей, дизайнер пользовательского интерфейса — создать прототип, а ведущий тестировщик — написать варианты тестирования. Сравнение представлений, созданных различными специалистами в ходе разнообразных исследований, помогает выявить несоответствия, неясности, допущения и упущения, которые трудно обнаружить, когда требования представлены в одном формате.

Определенные типы информации с помощью диаграмм удается передать гораздо эффективнее, чем с помощью текста. Изображения помогают преодолеть языковые барьеры и терминологические разногласия между членами команды. Бизнес-аналитику придется разъяснить остальным заинтересованным лицам назначение используемых моделей и применяемые условные обозначения. Существует большой выбор различных диаграмм и приемов моделирования, которые можно применять для наглядного представления требований. В этой главе рассказывается о нескольких приемах моделирования требований с иллюстрациями и ссылками на другие источники, где можно получить более подробную информацию.

Моделирование требований

Бизнес-аналитики могут надеяться найти единственный прием, позволяющий собирать всю информацию в одно целостное отображение требований к системе. К сожалению, подобной всеобъемлющей диаграммы не существует. В сущности, если бы можно было всю систему смоделировать в виде одной диаграммы, сама по себе она была бы столь же бесполезной, как и длинный список требований. Первоначальной целью структурированных систем анализа была полная замена классической функциональной спецификации графическими диаграммами и системами обозначений, более формальными, чем текстовые комментарии. Однако опыт показал, что модели анализа должны дополнять — а не заменять — спецификации требований на естественном языке. Тем не менее, подробность и точность письменных требований оказывается очень большим подспорьем для разработчиков и тестировщиков.

Визуальные модели требований могут помочь выявить отсутствующие, излишние и несовместимые требования. Ввиду ограничений краткосрочной памяти человека, анализ тысячи требований на предмет несоответствий, дублирования и посторонних требований практически невозможен. Дойдя до пятнадцатого требования вы скорее всего забудете первые. Вряд ли вы сможете найти все ошибки путем простого рецензирования текстовых требований.

К описанным в этой книге моделям визуального представления относятся:

- диаграммы потоков данных (data flow diagrams, DFD);
- диаграммы рабочих потоков, такие как диаграммы swimlane;

- диаграммы переходов состояний (state-transition diagrams, STD) и таблицы состояний;
- карты диалоговых окон;
- таблицы и деревья решений;
- таблицы событий и реакций;
- деревья функций (обсуждаются в главе 5)
- диаграммы вариантов использования (о которых рассказывалось в главе 8);
- диаграммы процессов (также обсуждались в главе 8);
- диаграммы «сущность-связь» (entity-relationship diagrams, ERD) (подробнее см. главу 13.)

Системы обозначений, представленные здесь, обеспечивают общий, стандартный для всей отрасли язык, которым должны пользоваться участники проекта. Изобретение собственной нотации моделирования несет с собой риск неправильной интерпретации — использовать стандартную нотацию все-таки безопаснее.

Эти модели годятся для разработки и изучения требований, а также для дизайна ПО. Будете ли вы использовать их для анализа или для дизайна, зависит от временных рамок и целей моделирования. Применение этих диаграмм для анализа требований позволит смоделировать предметную область или создать концептуальные представления новой системы. Они отображают логические аспекты компонентов данных предметной области, транзакции и преобразования, объекты реального мира и изменения состояния системы. Вы можете создавать модели на основании текстовых требований, чтобы представить их с различных точек зрения, или вывести детализированные функциональные требования из моделей высокого уровня, созданных на основе предоставленной пользователями информации. В процессе разработки модели демонстрируют, как вы намерены реализовать систему: базу данных, которую планируется создать, классы объектов, экземпляры которых будут создаваться, и модули кода, которые надо разработать. Так как в диаграммах анализа и дизайна используется одинаковая нотация, четко указывайте вид диаграммы: модель анализа (концепции) или модель дизайна (что планируется создать).

Приемы моделирования анализа, описанные в этой главе, поддерживаются различными серийными инструментами моделирования, управления требованиями и построения графических диаграмм, такими как Microsoft Visio. Использование этих инструментов обеспечивает некоторое преимущество перед обычными средствами рисования. Во-первых, они легко позволяют улучшить качество диаграмм при повторных просмотрах требований. Практически никогда не удается создать правильную модель с первого раза, поэтому итеративный подход можно назвать ключом к успеху при моделировании систем. Кроме того, инструменты автоматизированного проектирования ПО «знают» и применяют правила для каждого метода моделирования, который они поддерживают. Они способны определить синтаксические

ошибки и несоответствия, которые специалистам, проверяющим диаграммы, не всегда удается обнаружить. Средства управления требованиями позволяют отслеживать связи между требованиями и моделями. Некоторые инструменты связывают различные диаграммы друг с другом и с их связанными функциональными требованиями и требованиями к данным. Использование средства со стандартными обозначениями позволяет обеспечивать совместимость моделей друг с другом.

Мы слышали разные аргументы против использования моделей требований — от «Наша система слишком сложна для моделирования» до «У нас плотный график проекта и у нас нет времени на моделирование требований». Модель проще системы, которая моделируется. Если вы не в состоянии справиться со сложностью модели, то как вы собираетесь справляться со сложностью системы? Создание большинства моделей требует не больше времени, чем вы потратили бы на написание документа требований и анализа его на предмет проблем. Любое дополнительное время, потраченное на использование моделей анализа требований, более чем окупится за счет обнаружения всех ошибок требований. Модели или их части иногда можно повторно использовать между проектами или по крайней мере использовать их в качестве предварительной модели для выявления требований в следующем проекте.

От желания клиента к модели анализа

Тщательно слушая, как клиенты представляют свои требования, бизнес-аналитик может выделить ключевые слова, которые преобразуются в определенные элементы модели. В табл. 12-1 перечислены возможное соответствие значимых слов, употребляемых пользователями, компонентам модели, о которых рассказано далее в этой главе. По мере того как вы будете записывать пожелания пользователей в виде требований и моделей, вам придется отслеживать связи каждого компонента модели с конкретными пользовательскими требованиями.

Табл. 12-1. Привязка пожеланий клиента к компонентам модели анализа

Тип	Примеры	Компоненты модели анализа
Существительное	Люди, организации, системы ПО, элементы данных или существующие объекты	Внешние сущности, хранилища данных или потоки данных (диаграммы потоков данных) Действующие лица (диаграммы вариантов использования) Сущности или их атрибуты (диаграммы «сущность–связь») Дорожки (диаграммы swimlane) Объекты с состояниями (STD)

Табл. 12-1. (*окончание*)

Тип	Примеры	Компоненты модели анализа
Глагол	Действия, задачи, которые пользователь может выполнить, или события, которые могут произойти	Процессы (диаграммы потоков данных)
		Шаги процессов (диаграммы swim-lane)
		Варианты использования (диаграммы вариантов использования)
		Взаимосвязи (диаграммы «сущность–связь»)
		Переходы (диаграммы переходов состояний)
		Действия (диаграммы действий)
		События (таблица событий и реакций)
Условие	Выражения условной логики, такие как «если, то»	Решения (деревья решений, таблицы решений или диаграмма действий)
		Ветвление (диаграмма swimlane или диаграмма действий)

В этой книге я использовал в качестве учебного примера Chemical Tracking System. В следующем абзаце для этой системы описаны потребности пользователей, представленные сторонником продукта от класса «Химики». Значимые уникальные существительные выделены **полужирным** начертанием, глаголы — *курсивом*, а условия ***полужирным курсивом***; следите за этими ключевыми словами в моделях анализа, показанных далее в этой главе. На схемах некоторых моделей с целью иллюстрации показана информация, выходящая за рамки содержания следующего абзаца, тогда как на других моделях отображена только часть информации, представленной здесь:

«**Химик** или **сотрудник склада химикатов** может *разместить* **заказ** на один или несколько **химикатов**, ***если пользователь уполномочен делать такие заказы***. Заказ может быть *выполнен* или посредством *доставки* **контейнера** с химикатом, который числится в **инвентарной описи товаров** на складе химикатов, или посредством размещения **заказа** на химикат у стороннего **поставщика**. ***Если химикат относится к опасным***, он может быть доставлен, ***только если пользователь прошел соответствующее обучение***. **Сотрудник**, размещающий заказ, *подготавливая* заказ, должен иметь возможность *искать* в интерактивном режиме нужный химикат в **каталогах поставщиков**. Системе необходимо *отслеживать* **состояние** каждого заказа с момента его подготовки и до момента его выполнения или *отмены*. Ей также необходимо отслеживать **историю** каждого контейнера с химикатом с момента его *получения* **компанией** до его полного *расходования* или *утилизации*».

Внимание! Не стоит считать, что клиентам заранее известно, как читать модели анализа, однако не думайте, что они не смогут разобраться в них. Добавьте легенду и объясните цели и условные обозначения для каждой модели сторонникам продукта. Пройдите с ними пример модели, чтобы помочь им научиться читать тот или иной тип диаграмм.

Выбор правильного представления

Редкий случай, когда команде необходимо создать полный набор моделей анализа для всей системы. При моделировании сосредоточьтесь на наиболее сложных и опасных участках системы, а также на тех, где наиболее вероятны неясности и неопределенности. Элементы системы, от которых зависит ее безопасность и защита, а также элементы, влияющие на выполнение критически важных задач, можно смело назвать кандидатами на моделирование, так как последствие дефектов в них окажутся особенно тяжелыми. Также выбирайте модели, которые будете использовать вместе, чтобы гарантировать полноту всех моделей. Например, анализ объектов данных в диаграмме потоков данных помогает выявить отсутствующие сущности в диаграмме «сущность–связь». Анализ всех процессов в диаграмме потоков данных может оказаться полезным для построения нужных диаграмм swimlane. Далее в этой главе вы найдете информацию о том, какие модели дополняют друг друга.

В табл. 12-2, адаптированной из статьи Карла Вигера, содержатся рекомендации, какие приемы представления использовать в зависимости от того, какой тип информации вы пытаетесь показать, проанализировать или обнаружить. Даются дополнительные советы, какие модели требований создавать в зависимости от фаз и характеристик проекта, а также целевой аудитории моделей.

Табл. 12-2. Выбор наиболее подходящих способов представления

Отображение информации	Способы представления
Внешние интерфейсы системы	*Контекстная диаграмма* и *диаграмма вариантов использования* указывают внешние объекты, подключающиеся к системе. Контекстная диаграмма и *диаграмма потоков данных* иллюстрируют входную и выходную информацию системы на высоком уровне абстракции. *Карта экосистемы* указывает возможные системы, которые взаимодействуют, но также включает те, которые взаимодействуют не напрямую. *Диаграммы swimlane* показывают, что происходит в процессе взаиодействия между системами. Подробности внешних интерфейсов могут фиксироваться во входных и выходных *форматах файлов* и *макетов отчетов*. У продуктов, состоящих как из программных, так и аппаратных компонентов, часто есть *спецификации интерфейсов* с определениями атрибутов данных, обычно в форме API-интерфейса или конкретных входных и выходных сигналах аппаратного устройства

(*см. след. стр.*)

Табл. 12-2. (*продолжение*)

Отображение информации	Способы представления
Поток бизнес-процессов	*Диаграмма потоков данных* верхнего уровня представляет на высоком уровне абстракции, как бизнес-процесс работает с данными. *Диаграмма swimlane* показывают роли, участвующие в выполнении различных этапов в потоке бизнес-процессов. Уточняющие уровни *диаграмм потоков данных* или *диаграмм swimlane* могут очень детально представлять потоки бизнес-процессов. Аналогично *диаграммы потоков* и *диаграммы действий* могут использоваться как на высоком, так и низком уровне абстракции, хотя чаще всего они используются для определения деталей процесса
Определения данных и связи объектов данных	*Диаграмма «сущность–связь»* показывает логические отношения между объектами данных (сущностями). *Диаграмма классов* показывает логические связи между классами объектов и связанными с ними данными. *Словарь данных* содержит подробные определения структур данных и отдельных элементов данных. Сложные объекты данных последовательно разбиваются на составляющие элементы данных
Состояния системы и объектов	*Диаграммы переходов состояния* и *таблицы состояния* дают представление высокого уровня абстракции возможных состояний системы или объекта и изменений между состояниями, которые могут происходить в определенных обстоятельствах. Эти модели полезны, когда определенные объекты могут работать (и изменять свое состояние) во многих вариантах использования. Некоторые аналитики создают *таблицу событий и реакций* в качестве инструмента определения границ продукта на основе внешних событий. В этой таблице можно также указать отдельные функциональные требования путем уточнения, как система должна вести себя в ответ на каждую комбинацию внешних событий и состояния системы. *Функциональные требования* предоставляют подробности того, как в точности поведение системы и пользователя должно приводить к изменениям состояний
Сложная логика	*Дерево решений* показывает возможные результаты связанных решений и условий. *Таблица решений* определяет уникальные функциональные требования, связанные с различными комбинациями результатов true и false для наборов решений или условий

Табл. 12-2. (*окончание*)

Отображение информации	Способы представления
Пользовательские интерфейсы	*Карта диалоговых окон* предоставляет высокоуровневое представление предлагаемого или фактического пользовательского интерфейса и показывает разные элементы отображения и пути навигации между ними. *Раскадровки* и *бумажные прототипы* дополняют карту диалоговых окон, показывая, что должно содержать каждое окно, но не предоставляя точных деталей. *Модели «отображение-действие-реакция»* описывают отображение и поведение требований на каждом экране. *Подробные макеты экрана* и *бумажные прототипы* показывают в точности, как должны выглядеть элементы интерфейса. *Определения полей данных* и *описания элементов управления пользовательским интерфейсом* предоставляют дополнительные подробности
Описания задач пользователей	*Пользовательские истории*, *спецификации сценариев и вариантов использования* описывают задачи пользователей с различной степенью детализации. *Диаграммы swimlane* иллюстрируют бизнес-процесс или взаимодействие между многими действующими лицами и системой. *Блок-схемы* и *диаграммы действий* наглядно отображают направление диалога варианта использования и разветвляются на альтернативные направления и исключения. *Функциональные требования* предоставляют подробные описания, как система и пользователь будут взаимодействовать для достижения полезных результатов. *Варианты тестирования* предоставляют альтернативное представление низкого уровня абстракции, описывающее какое точно поведение следует ожидать от системы при определенных условиях входных данных, состоянии системы и действиях
Нефункциональные требования (атрибуты качества, ограничения)	Атрибуты качества и ограничения обычно пишутся в форме текста *на естественном языке*, но при этом часто не хватает точности и полноты. В главе 14 описывается исчерпывающая методика для точного определения нефункциональных требований, которая называется *Planguage* (Gilb, 2005)

Диаграмма потоков данных

Диаграмма потоков данных (*data flow diagram,* DFD) — основной инструмент структурного анализа (DeMarco, 1979; Robertson и Robertson, 1994). Она позволяет определять процессы преобразования системы, совокупность (хранилище) данных или материалов, которыми система управляет, и потоки данных или материалов между процессами, хранилищами и внешним миром.

При моделировании потоков данных для системного анализа используется прием разложения функций, при котором сложные проблемы раскладываются на составляющие, размещаемые по нарастающей по уровням детализации. Этот подход отлично подходит для систем обработки транзакций и других приложений с большой нагрузкой на функциональность. Добавление элементов управления потоками позволило применять диаграммы потоков данных для моделирования систем реального времени (Hatley, Hruschka и Pirbhai, 2000).

Диаграммы потоков данных предоставляют общее представление о том, как данные перемещаются в системе, — другие модели показывают это не так хорошо. Различные люди и системы выполняют процессы, в которых используются, изменяются и производятся данные, поэтому один вариант использования или диаграмма swimlane не в состоянии показать весь цикл жизни того или иного элемента данных. Также многие элементы данных могут собираться вместе и преобразовываться процессом (например, содержимое корзины для покупок, информация о доставке и информация об оплате преобразуются в объект-заказ). И это сложно показать в других моделях. Однако диаграммы потоков данных не могут служить единой методикой моделирования. Подробности преобразования данных в процессе лучше видны в диаграммах вариантов использования или диаграммах swimlane.

Битти и Чен (Beatty и Chen, 2012) дают советы по созданию и использованию диаграмм потоков данных для анализа требований. Это средство часто используется при интервьюировании клиентов, потому что очень просто набросать диаграмму потоков данных на доске при обсуждении того, как работает бизнес пользователя. Диаграммы потоков данных могут применяться в качестве средства обнаружения пропущенных требований к данным. Данные, «перетекающие» между процессами, хранилищами данных и внешними сущностями следует также моделировать в диаграммах «сущность-связь» и описывать в словаре данных. Также диаграммы потоков данных обеспечивают контекст для функциональных требований в смысле того, как пользователь выполняет определенные задачи, например заказ химиката.

Диаграммы потока данных могут представлять системы на самых разных уровнях абстракции: диаграммы высокого уровня предоставляют целостный, панорамный вид компонентов данных и обработки в многоэтапном процессе, дополняющем точное, детальное представление, приведенное в функциональных требованиях. Контекстная диаграмма на рис. 5-6 представляет наивысший уровень абстракции диаграммы потока данных. Контекстная диаграмма представляет всю систему как единый процесс в виде черного ящика, изображенного в виде кружка. На ней также показаны *внешние сущности*, или оконечные объекты (terminators), подключенные к системе и данным или материальным потокам между системами и оконечными объектами. Потоки на контекстной диаграмме часто представляют сложные структуры данных, определенные в словаре данных.

Вы можете конкретизировать контекстную диаграмму до уровня 0 (верхний уровень додели потоков данных), который разбивает систему на ее основные процессы. На рис. 12-1 показан (несколько упрощенный) уровень 0 диаграммы потоков данных для системы Chemical Tracking System. В этой модели используется нотация Йодона-ДеМарко. Есть альтернативные нотации, в которых применяются немного отличающиеся символы.

Единый кружок, обозначающий Chemical Tracking System на контекстной диаграмме, был разделен на семь основных процессов (кружков). Как и на контекстной диаграмме, оконечные объекты показаны в прямоугольниками. Все потоки данных (стрелки), присутствовавшие на контекстной диаграмме, отражены на уровне 0 диаграммы потоков данных. Кроме того, парами параллельных линий здесь обозначены *хранилища данных*, которые относятся ко внутренней части системы и, следовательно, не показаны на контекстной диаграмме. Поток от кружка к хранилищу, указывает, что данные были помещены в хранилище, поток, выходящий из хранилища, обозначает операцию чтения, а двунаправленная стрелка между хранилищем и кружком — операцию обновления.

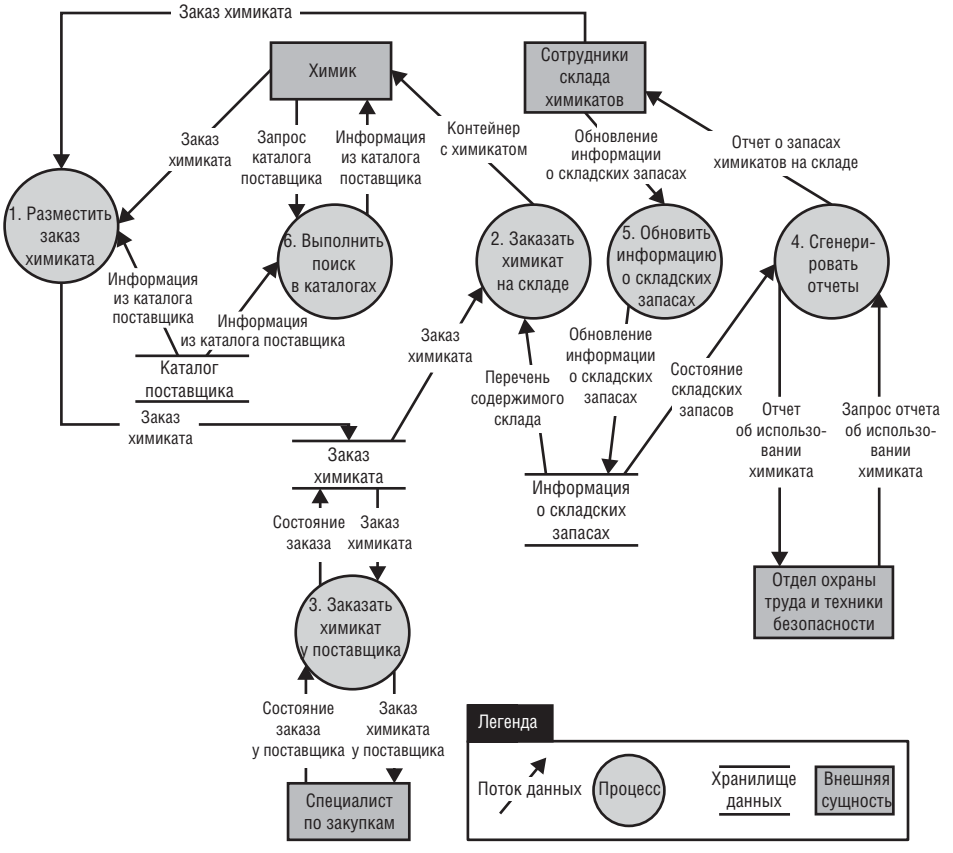


Рис. 12-1. Уровень 0 диаграммы потока данных для Chemical Tracking System (сокращенная версия)

Каждый процесс, изображенный в виде отдельного кружка на уровне 0 диаграммы, можно расписать более подробно с помощью отдельной диаграммы, изобразив на ней всю функциональность этого процесса. Бизнес-аналитик продолжает последовательные уточнения до тех пор, пока диаграммы нижних уровней не будут содержать только простейшие операции, которые можно ясно представить в виде теста, диаграммы swimlane или диаграммы действий. Функциональные требования в спецификации требований к ПО точно определят, что происходит в ходе каждого простейшего процесса. Диаграммы потока данных каждого уровня должны быть сбалансированы и согласованы с расположенным выше уровнем так, чтобы все входящие и исходящие потоки на дочерней диаграмме соответствовали аналогичным на родительской диаграмме. Сложные структуры данных в диаграммах высоких уровней могут быть разделены на составные элементы, как определено в словаре данных, на диаграммах потока данных нижних уровней.

На первый взгляд схема на рис. 12-1 выглядит сложной. Однако если вы изучите ближайшее окружение любого процесса, то увидите элементы данных, которые он потребляет и производит, а также их исходные точки и точки назначения. Чтобы увидеть, как именно в процессе используются элементы данных, вам понадобится либо нарисовать более подробную дочернюю диаграмму потоков данных или сослаться на функциональные требования для этой части системы.

Ниже приведены некоторые правила для рисования диаграмм потока данных. Не все придерживаются одних и тех же правил (например, некоторые бизнес-аналитики показывают оконечные сущности только на контекстных диаграммах), однако мы считаем их полезными. Использование моделей для улучшения коммуникации между участниками проекта важнее догматического соблюдения этих правил.

- Процессы взаимодействуют через хранилища данных, а не посредством прямых потоков от одного процесса к другому. Точно так же данные не могут передаваться напрямую из одного хранилища в другое, они должны пройти через процесс (кружок).
- Не пытайтесь передать последовательность этапов процесса с помощью диаграммы потоков данных.
- Называйте процессы как короткое действие: глагол + объект (например, «Создать инвентарный отчет»). Используйте названия, понятные клиентам и употребляемые в деловой или предметной области.
- Присваивайте процессам уникальные номера согласно иерархии. На уровне 0 диаграммы, пронумеруйте каждый процесс целыми числами. Если вы создадите дочернюю диаграмму потоков данных для процесса 3, пронумеруйте процессы в ней как 3.1, 3.2 и т.д.
- Не показывайте на одной диаграмме более 8-10 процессов, в противном случае ее будет трудно рисовать, изменять и понимать. Если у вас больше десяти процессов, создайте еще один уровень абстракции, сгруппировав связанные процессы в процесс более высокого уровня.

- Кружки с только входящими или только исходящими потоками вызывают подозрение. Корректный процесс, изображаемый кружком на диаграмме потоков данных, как правило, отличается наличием и входящих, и исходящих потоков.

Рецензируя диаграмму потоков данных, представители клиентов должны убедиться, что на ней отображены все известные и относящиеся к делу процессы и что у них нет отсутствующих или ненужных входящих или исходящих потоков. При рецензировании диаграммы потоков данных часто выявляются не распознанные ранее классы пользователей, бизнес-процессы и связи с другими системами.

Моделирование задач, а не ПО

Как-то я в качестве представителя от ИТ-отдела работал в команде, которая занималась реинжинирингом бизнес-процессов. Наша задача заключалась в сокращении в 10 раз времени доступности нового химиката для использования в продукте. В команду были включены представители следующих подразделений компании, задействованных в коммерциализации химиката:

- химик, который первым синтезировал новое вещество;
- химик-технолог, разрабатывающий процесс промышленного производства вещества в больших объемах;
- химик-аналитик, разрабатывающий методы проверки чистоты вещества;
- патентные поверенные, которые занимаются патентной защитой изобретения;
- специалисты отдела охраны труда и здоровья, получающие разрешение правительства на использование химиката в потребительских товарах.

После того как мы разработали новый процесс, который по нашему мнению должен был необыкновенно ускорить коммерческое применение химиката, и смоделировали его в диаграмме swimlane, я опросил членов команды, ответственных за каждый этап процесса. Я задал каждому два вопроса: «Какая информация вам нужна для выполнения этого этапа?» и «Какую информацию, которая появится в результате выполнения этого этапа, следует сохранить?». Собрав и сравнив ответы, я выявил те этапы, которым нужна информация, но ее никто не поставлял, а также этапы, выходная информация которых не была нужна никому. Мы разрешили эти проблемы.

Затем я нарисовал диаграмму потоков данных, чтобы проиллюстрировать новый процесс коммерциализации химиката и диаграмму «сущность–связь» (см. главу 13) для моделирования связей данных. Все элементы данных были определены в словаре данных (см. главу 13). Такие модели анализа будут полезны в качестве инструментов коммуникации, помогающих членам команды выработать общее понимание нового про-

цесса. Кроме того, модели представляют собой прекрасную точку отсчета для разработки требований для приложений и определения их границ.

Диаграммы swimlane

Диаграммы swimlane — один из способов представления шагов, из которых состоит бизнес-процесс или операции предлагаемой программной системы. Это вариант блок-схемы, разделенной на видимые подкомпоненты, которые называются *дорожками* (lanes). Дорожки могут представлять разные системы или действующих лиц, которые выполняют шаги процесса. Диаграммы swimlane чаще всего применяются для отображения бизнес-процессов, последовательности процессов или взаимодействия системы и пользователя. Они похожи на UML-диаграммы действий. Диаграммы swimlane иногда называют межфункциональными диаграммами.

Они могут показать, что происходит внутри кружка процесса в диаграмме потоков данных. Они помогают связать вместе функциональные требования, которые позволяют пользователям выполнять определенные задачи. Их можно также применять для выполнения детального анализа в процессе выявления требований, относящихся к каждому шагу процесса (Beatty и Chen, 2012).

Диаграмма swimlane — одна из самых простых диаграмм для понимания заинтересованными лицами, потому что нотация проста и широко используется. Создание предварительной версии бизнес-процессов в виде диаграммы swimlane может стать хорошей отправной точкой дискуссий по выявлению требований, как описано в главе 24. Диаграммы swimlane могут содержать дополнительные фигуры, но самые популярные:

- **шаги процесса** в виде прямоугольников;
- **переходы между шагами процесса**, показанные как стрелки, соединяющие пары прямоугольников;
- **решения** — ромбы с несколькими исходящими ветками. Варианты решений показаны в виде текстовых надписей на стрелках, выходящих из ромба;
- **дорожки** (swimlane), разбивающие процесс, показанные как горизонтальные или вертикальные линии. Дорожки, как правило, отвечают ролям, подразделениям или системам. Они показывают, кто или что выполняет шаги на данной дорожке.

На рис. 12-2 показана частичная диаграмма swimlane системы Chemical Tracking System. В данном примере дорожки соответствуют ролям или подразделениям, показывая, какие шаги выполняет каждая группа в бизнес-процессе заказа химиката у поставщика. Определение функциональных требований можно начать с первого прямоугольника «Создание заказа химиката», после чего подумать, какую функциональность должна поддерживать система, чтобы поддержать этот шаг, а также каковы требования к данным для «заказа химиката». Следующий шаг «Проверка и одобрение счета» мо-

жет вызвать выявление требование, в котором надо выяснить, что означает обработка счета. Как получают счет? Каков у него формат? Обрабатывается ли счет вручную или часть работы берет на себя система? Передаются ли данные счета в другие системы?

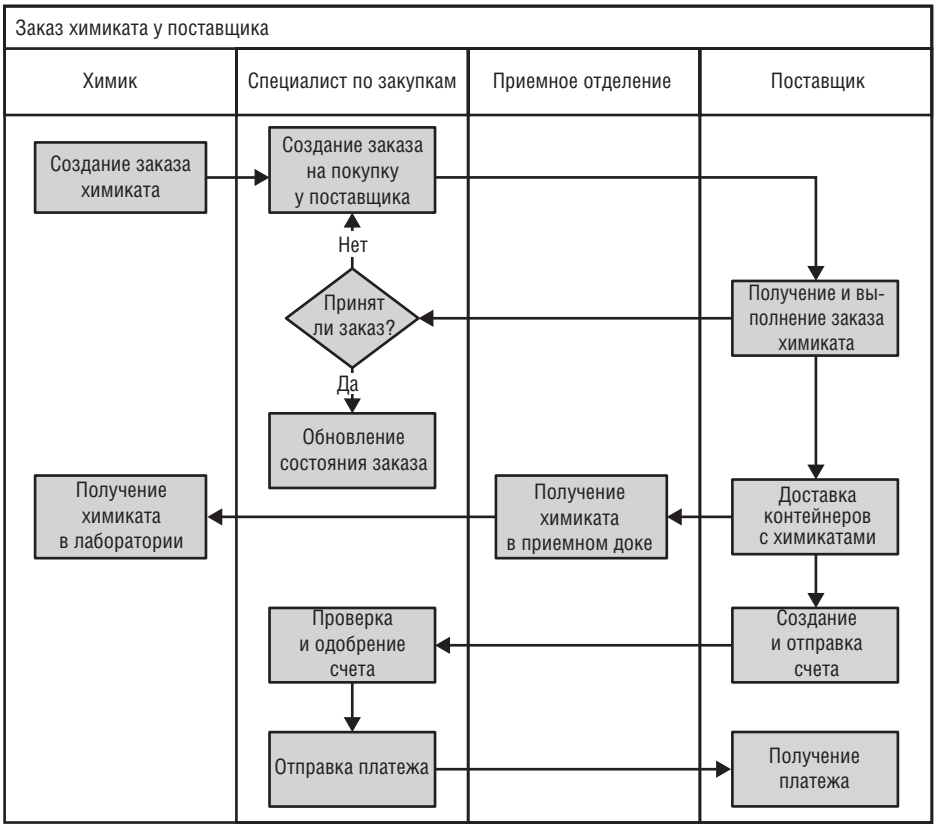


Рис. 12-2. Диаграмма swimlane процесса в системе Chemical Tracking System (частичная)

Весь бизнес-процесс может полностью и не укладываться в границы программной системы. Обратите внимание, что Приемное отделение представляет собой часть процесса, но его нет в контекстной диаграмме и диаграмме потоков данных, потому что оно никогда не взаимодействует с Chemical Tracking System напрямую. Вместе с тем, анализ карты экосистемы на рис. 5-7 (см. главу 5) позволил команде понять, что в бизнес-процессе участвует Приемное отделение. В команде также проверили входные и выходные данные кружка на диаграмме потоков данных (процесс 3 на рис. 12-1), чтобы убедиться, что обе модели потребляют и производят одинаковые данные, а также исправили все обнаруженные ошибки. Это иллюстрирует мощь моделирования, создания множественных представлений с использованием разных мыслительных процессов для получения более глубокого понимания создаваемой системы.

Диаграмма переходов состояний и таблица состояний

Для любых систем ПО необходимо учитывать комбинацию функционального поведения, особенности обработки данных и изменения состояния. Состояний, в которых в каждый момент времени могут находиться системы реального времени и приложения, управляющие процессами, немного. Изменение состояния происходит, только когда удовлетворяются четко определенные критерии, такие как получение определенного сигнала на входе при определенных условиях. Примером может служить перекресток магистрали с установленными датчиками движения, защищенной полосой поворота, а также разметкой и сигналами для пешеходного перехода. Многие информационные системы имеют дело с бизнес-объектами — заказы на покупку, счета-фактуры, товарно-материальные ценности и т. п. — для жизненных циклов которых возможно несколько различных состояний, или статусов.

При описании сложных изменений состояний на естественном языке высока вероятность упустить из внимания разрешенное изменение состояния или появление запрещенного изменения. В зависимости от структуры спецификации требований к ПО, требования, относящиеся к поведению состояния механизма, могут быть в ней разобщены. В этом случае весьма трудно понять поведение системы.

Диаграмма переходов состояний (state-transition diagram) позволяет получить лаконичное, полное и недвусмысленное представление о состояниях объекта или системы. Она наглядно показывает возможные переходы между состояниями. Связанный с этой моделью прием — диаграмма состояний — включен в обладающий более богатым набором условных обозначений унифицированный язык моделирования (Unified Modeling Language, UML), который моделирует состояния объекта в течение его жизненного цикла (Ambler, 2005). Диаграмма переходов состояний содержит три типа элементов:

- возможные состояния системы — показаны в виде прямоугольников. В некоторых нотациях для представления состояния используются кружки (Beatty и Chen, 2012). Можно использовать кружки или прямоугольники — просто выбрав что-то одно, нужно придерживаться выбранной нотации;
- разрешенные состояния, или *переходы*, — показаны в виде стрелок, соединяющих пары прямоугольников;
- события или условия, вызывающие каждый переход, — показаны в виде текстовых пояснений для каждой стрелки перехода. Текст может пояснять и событие, и соответствующую реакцию системы.

Диаграмма переходов состояний для объекта, имеющего определенный жизненный цикл, будет иметь одно или более конечных состояний, которые представляют конечные состояния, которые могут быть у объекта. У конечных состояний есть входящие стрелки переходов, но нет исходящих. Клиенты

могут читать диаграммы переходов состояний после небольшого инструктажа по нотации — ведь это всего лишь прямоугольники и стрелки.

Вспомните из главы 8, что основная функция Chemical Tracking System — позволить действующим лицам, которые называются «Сотрудниками, разместившими заказ на химикат», размещать заказы химикатов, которые могут выполняться либо складскими работниками (если химикат есть на складе), либо сторонним поставщиком (для этого ему надо отправить заказ). Каждый заказ проходит через несколько состояний с момента его создания до момента либо его выполнения, либо отмены (два конечных состояния). Таким образом, диаграмма переходов состояний моделирует жизненный цикл заказа химиката, как показано на рис. 12-3.

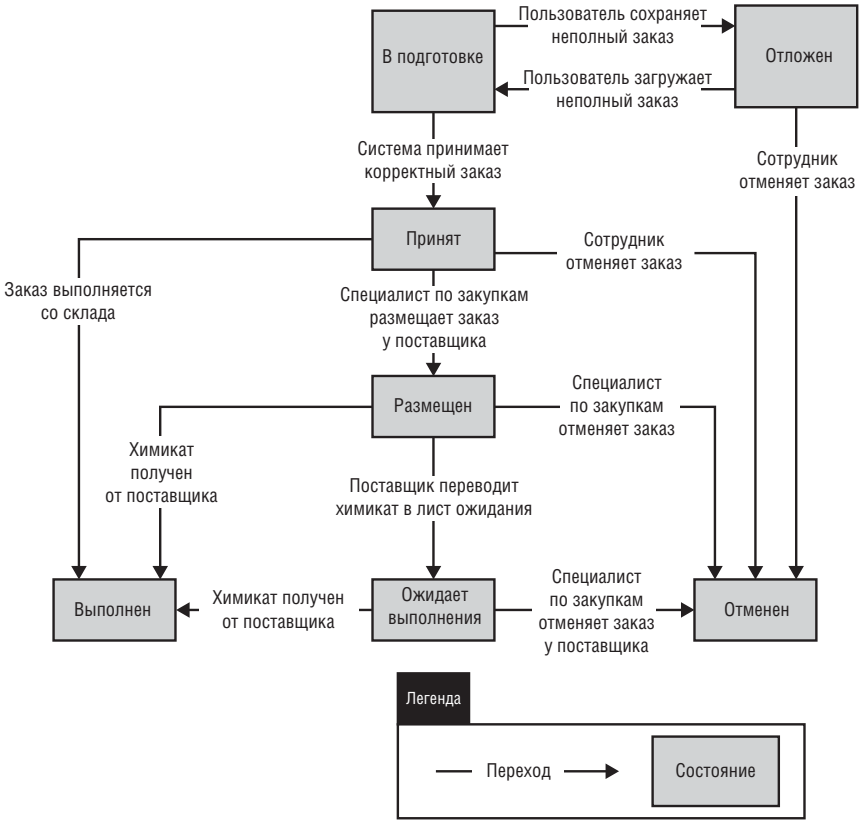


Рис. 12-3. Частичная диаграмма переходов состояний для заказа химиката в Chemical Tracking System

Эта диаграмма переходов состояний показывает, что заказ может находиться в одном из следующих семи возможных состояний:

- **В подготовке** Сотрудник создает новый заказ, инициировав эту функцию из другой части системы;
- **Отложен** Сотрудник сохраняет частичный заказ для завершения его в будущем, не передавая его в систему и не отменяя операцию заказа;

- **Принят** Сотрудник отправляет готовый заказ химиката, и система принимает его к исполнению;
- **Размещен** Заказ должен быть удовлетворен сторонним поставщиком, а специалист по закупкам разместил заказ у продавца;
- **Выполнен** Заказ удовлетворен: контейнер с химикатом поставлен либо со склада химикатов, либо от поставщика;
- **Ожидает выполнения** У продавца не оказалось химиката в наличии, и он уведомил покупателя, что заказ отложен до поставки в будущем;
- **Отменен** Сотрудник отменил принятый системой заказ до того, как тот был выполнен, или специалист по закупкам отменил заказ у продавца, до того как тот был выполнен или пока ожидал выполнения.

Когда представители пользователей Chemical Tracking System просмотрели диаграмму переходов состояний для заказа химиката, они определили, что одно состояние не нужно, другое важное состояние отсутствует, и указали два неправильных перехода. При изучении соответствующих функциональных требований эти ошибки никто из них не заметил. В этом и заключается важность представления информации о требованиях на нескольких уровнях абстракции. Зачастую проблему легче обнаружить, если идти назад от наиболее детализированного уровня и рассматривать большую картину, которую предоставляют модели анализа. Однако диаграмма переходов состояний не дает уровень деталей, достаточный разработчику для создания ПО. Следовательно, в спецификацию требований к ПО для Chemical Tracking System были включены функциональные требования, связанные с обработкой заказа химиката и возможными изменениями его состояния.

Таблица состояний показывает все возможные переходы между состояниями в виде матрицы. Бизнес-аналитик может использовать таблицы состояний для проверки, что все переходы обнаружены, путем анализа всех ячеек матрицы. Все состояния выписаны в первом столбце и повторены в первой строке. Ячейки содержат информацию о том, возможны ли переходы из состояния в левом столбце в состояние, указанное в верхней строке, а также указывает событие перехода между состояниями. На рис. 12-4 показана таблица состояний, соответствующая диаграмме переходов состояний на рис. 12-3. Эти две диаграммы содержат одну и ту же информацию, но табличный формат помогает не пропустить никаких переходов, а формат диаграммы помогает заинтересованным лицам наглядно увидеть возможные последовательности переходов. Может не требоваться создавать обе модели. Однако создав одну модель, скомпоновать вторую просто, если вам нужно проанализировать изменения состояний с двух сторон. Две строки на рис. 12-4, в которых содержатся только значения «Нет», являются конечными состояниями; заказ в состоянии «Выполнен» или «Отменен» не может перейти в другое состояние.

Диаграмма переходов состояний и таблица состояний предоставляют высокоуровневую картину, охватывающую много вариантов использования и пользовательских историй, каждая из которых может выполнять переход от

одного состояния к другому. В моделях состояний не показаны детали обработки, выполняемой системой, а только возможные изменения состояний, возникающие в результате этих процессов. Они помогают разработчику понять ожидаемое поведение системы. Модели способствуют тестированию на ранних этапах, потому что тестировщики могут вывести тесты на основании диаграммы переходов состояний, которая охватывает все допустимые пути переходов. Обе модели хороши для проверки того, все ли необходимые состояния и переходы состояний корректно и полно описаны в функциональных требованиях.

	В подготовке	**Отложен**	**Принят**	**Размещен**	**Ожидает выполнения**	**Выполнен**	**Отменен**
В подготовке	нет	Пользователь сохраняет неполный заказ	Система принимает корректный заказ	нет	нет	нет	нет
Отложен	Пользователь открывает неполный заказ	нет	нет	нет	нет	нет	нет
Принят	нет	нет	нет	Специалист по закупкам размещает заказ у поставщика	нет	Заказ выполняется со склада	Сотрудник отменяет заказ
Размещен	нет	нет	нет	нет	Поставщик переводит химикат в лист ожидания	chemical received from vendor	Специалист по закупкам отменяет заказ
Ожидает выполнения	нет	нет	нет	нет	нет	chemical received from vendor	Специалист по закупкам отменяет заказ
Выполнен	нет	нет	нет	нет	нет	нет	нет
Отменен	нет	нет	нет	нет	нет	нет	нет

Рис. 12-4. Таблица состояний заказа химиката в системе Chemical Tracking System

Карта диалоговых окон

Карта диалоговых окон представляет дизайн пользовательского интерфейса на высоком уровне абстракции. На ней показаны элементы диалоговых окон в системе и ссылки навигации между ними, но не показан подробный дизайн экрана. Пользовательский интерфейс также можно рассматривать как набор изменений состояний. Только один элемент диалогового окна (такой, как меню, рабочая область, диалоговое окно, командная строка или дисплей сенсорного экрана) доступен в определенный момент времени для ввода информации пользователем. Пользователь может перейти к другим определенным элементам диалогового окна, связанным с действием, которые он выполняет в активной области ввода. Количество возможных путей навигации в сложном графическом интерфейсе велико, однако оно конечно и, как правило, все возможности известны. Карта диалоговых окон на самом деле представляет собой всего лишь пользовательский интерфейс, смоделированный в диаграммы переходов между состояниями (Wasserman, 1985; Wiegers, 1996). Constantine и Lockwood (1999) описывают похожий прием —

карта перемещений (navigation map), которую отличает более богатый набор условных обозначений для представления различных типов элементов взаимодействия и контекстных переходов. *Поток пользовательского интерфейса* (user interface flow) похож карту диалоговых окон, но показывает пути навигации между окнами пользовательского интерфейса в формате диаграммы swimlane (Beatty и Chen, 2012).

Карта диалоговых окон позволяет рассмотреть возможные концепции пользовательского интерфейса с учетом вашего понимания требований. Пользователям и разработчикам стоит изучить карту диалоговых окон, чтобы выработать общее представление того, как пользователь может взаимодействовать с системой для выполнения задачи. Карты диалоговых окон также полезны при моделировании визуальной архитектуры веб-сайта. Ссылки для перемещений, которые вы включаете в веб-сайт, на картах диалоговых окон изображаются в виде переходов. Естественно, у пользователя есть дополнительные возможности навигации с помощью кнопок «Вперед» и «Назад», а также поле ввода URL-адреса, но карты диалоговых окон этого не показывают. Карты диалоговых окон связаны с системными раскадровками, в которые также включено краткое описание назначения каждого экрана (Leffingwell и Widrig, 2000).

Карты диалоговых окон отражают сущность взаимодействий системы и пользователя и потоки задач без тормозящих работу команды деталей макета экрана. С помощью такой карты пользователи могут отследить отсутствующие, неправильные или ненужные переходы и, следовательно, отсутствующие, неправильные или ненужные требования. Абстрактная концептуальная карта диалоговых окон, разработанная в ходе анализа требований, становится руководством для подробного дизайна пользовательского интерфейса.

Так же, как и обычные диаграммы перехода состояния, карта диалоговых окон показывает каждый элемент диалогового окна как состояние (прямоугольник) и каждую допустимую возможность перемещения как переход (стрелка). Условие, инициирующее перемещение по пользовательскому интерфейсу, показано в виде текстового ярлыка на стрелке перехода. Существует несколько типов инициализирующих условий:

- действие пользователя, например нажатие функциональной клавиши, щелчок гиперссылки или кнопки диалогового окна или жест на сенсорном экране;
- значение данных, такое как недостоверная информация, в результате чего появляется сообщение об ошибке;
- системное условие, например отсутствие бумаги в принтере;
- некоторые комбинации вышеперечисленных элементов, например ввод номера элемента меню и нажатие клавиши Enter.

Карты диалоговых окон слегка напоминают диаграммы потоков, но у них другое назначение. Диаграммы потока ясно показывают этапы процесса и

точки решений, но не пользовательский интерфейс. В отличие от них, на карте диалоговых окон *не отображается* процесс, выполняющийся по линиям перехода, которые соединяют диалоговые окна. Решения ветвления (обычно это выбор пользователя) скрыто за окнами, которые показаны на карте диалоговых окон в виде прямоугольников, а условия, в результате которых отображается тот или другой экран, описаны над стрелками переходов.

Чтобы упростить карту диалоговых окон, пропустите глобальные функции, такие, как нажатие клавиши F1 для вызова справки для каждого элемента диалогового окна. В разделе спецификации требований к ПО, посвященному пользовательскому интерфейсу, должно быть указано, что эта функциональность будет доступна, но демонстрация множества экранов справки на карте диалоговых окон вносит в модель беспорядок, а пользы от этого немного. Точно так же при моделировании веб-сайта не нужно включать стандартные для каждой страницы ссылки перемещения. Вы также можете опустить переходы, реализующие последовательность перемещений по веб-странице в обратном направлении, которые выполняются по щелчку кнопки «Назад» браузера.

Карта диалоговых окон — это прекрасный способ представить взаимодействия действующего лица и системы, описываемые вариантом использования. Карта диалоговых окон позволяет отобразить альтернативные направления в виде ответвлений от нормального направления. Я обнаружил, что наброски фрагментов карты диалоговых окон на доске оказались особенно полезны на семинарах по сбору информации, касающейся вариантов использования, когда участники изучали последовательность действий действующих лиц и реакций системы при выполнении задачи. Уже готовые варианты использования и потоки процессов сравните с картами диалоговых окон, чтобы убедиться, что все функции, необходимые для выполнения шагов, доступны при навигации пользовательского интерфейса.

В главе 8 описан вариант использования Chemical Tracking System под названием «Заказ химиката». Нормальное направление развития этого варианта использования подразумевает заказ контейнера с химикатом со склада. Альтернативное направление — заказ химиката у поставщика. Пользователь, размещающий заказ, хотел бы иметь возможность просматривать историю доступных контейнеров с этим химикатом, прежде чем выбрать один из них. На рис. 12-5 показана карта диалоговых окон для этого достаточно сложного варианта использования. Входной точкой этой карты диалоговых окон является линия перехода, начинающаяся с черного кружка «запросить размещение заказа». Пользователь будет попадать в эту часть пользовательского интерфейса приложения из другой части интерфейса. Точки выхода из карты диалоговых окон для возвращения в другую часть пользовательского интерфейса представляют собой линии перехода, заканчивающиеся черных кружках в четном кольце — «отменить весь заказ» и «ОК, выйти из функции заказа».

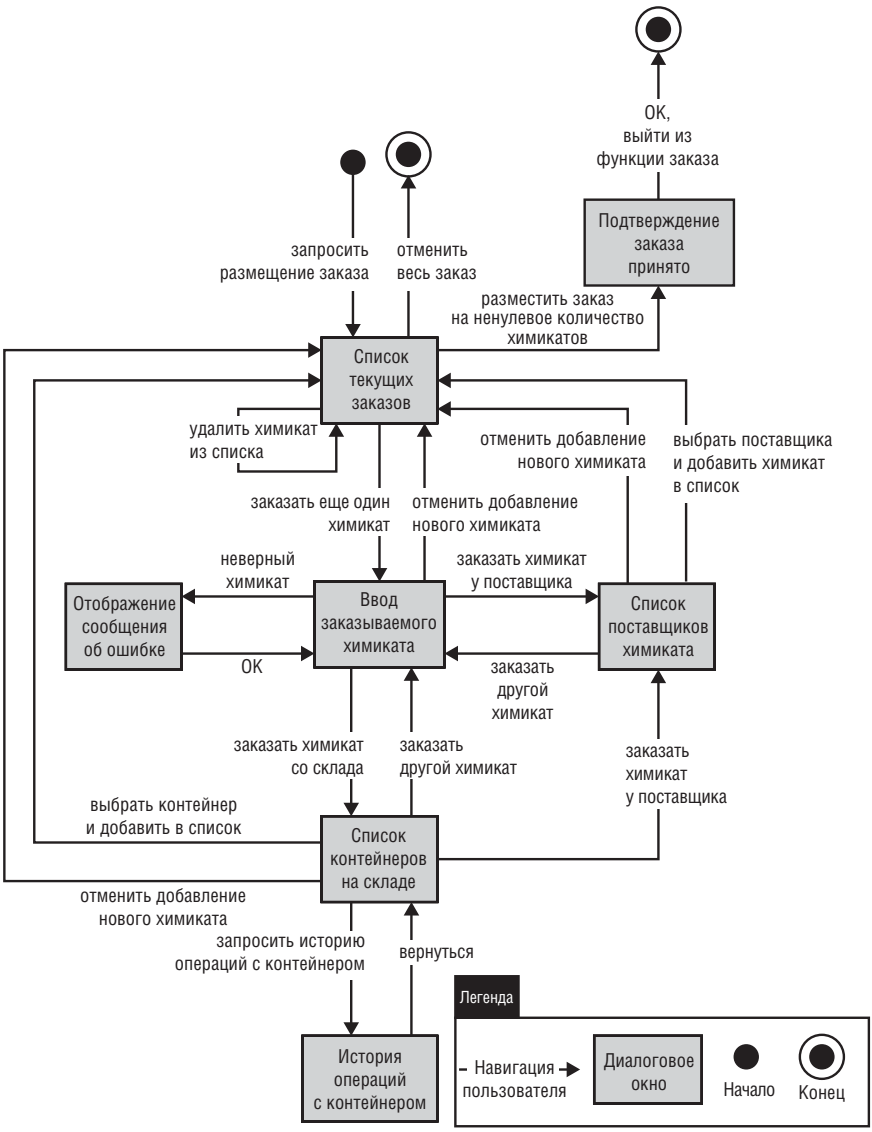


Рис. 12-5. Карта диалоговых окон для варианта использования «Заказ химиката» в системе Chemical Tracking System

На первый взгляд эта диаграмма может показаться довольно запутанной, но в ней довольно легко разобраться, если попробовать за один раз отслеживать один прямоугольник и одну линию. Пользователь инициирует этот вариант использования, размещая заказ химиката из меню Chemical Tracking System. В карте диалоговых окон это действие пользователя, обозначенное стрелкой в верхней левой части карты, направленной к прямоугольнику с названием «Список текущих заказов». Этот прямоугольник представляет основное рабочее пространство для такого варианта использования, т. е. спи-

сок химикатов в текущем заказе пользователя. Стрелки, исходящие из этого прямоугольника на карте диалогов показывают все возможные навигации — и, следовательно, функциональность — доступные пользователю в данном контексте:

- отменить весь заказ;
- разместить заказ, если он содержит как минимум один химикат;
- добавить в список заказов еще один химикат;
- удалить химикат из списка.

В последней операции, удалении химиката, не участвует какой-либо еще элемент диалогового окна, при этом просто обновляется список текущих заказов после того, как пользователь вносит изменения.

По мере перемещения по этой карте диалоговых окон вы увидите элементы, отражающие остальные элементы варианта использования «Заказ химиката»:

- один путь для заказа химиката у поставщика;
- другой путь для выполнения заказа через склад химикатов;
- еще один возможный путь — просмотр истории операций с контейнером, хранящимся на складе химикатов;
- отображение сообщения об ошибке для обработки ввода неправильного идентификатора химиката или других возможных ошибочных условий.

Некоторые переходы на карте диалоговых окон позволяют пользователю откатить операцию. Пользователей раздражает, если они передумали, а им все-таки приходится завершать задачу. Карты диалоговых окон позволяют облегчить и упростить работу, предлагая функции отката и отмены в стратегических точках.

Пользователь, изучающий карту диалоговых окон, может обнаружить недостающее требование. Например, осторожный пользователь может захотеть подтвердить операцию, которая отменяет весь заказ, чтобы избежать непроизвольной потери данных. Дешевле добавить такую функцию на этапе анализа, чем встраивать ее в уже законченный продукт. Поскольку карта диалоговых окон представляет концептуальные элементы, участвующие во взаимодействии пользователя и системы, не пытайтесь зафиксировать все детали дизайна пользовательского интерфейса на этапе работы над требованиями. Лучшее применение этих моделей — помочь всем заинтересованным лицам проекта выработать общее понимание предполагаемой функциональности системы.

Таблицы и деревья решений

Часто система ПО управляется сложной логикой, учитывающей различные комбинации условий, результатом которых является различное поведение системы. Например, если водитель нажимает кнопку ускорения в системе круиз-контроля автомобиля и автомобиль в данный момент двигается под

управлением круиз-контроля, то система увеличит скорость автомобиля, в противном случае команда игнорируется. Для спецификации требований к ПО необходимы функциональные требования, в которых будет описано, что должна делать система при всех комбинациях условий. Однако условие легко пропустить, в результате чего пропускается и требование. Эти пробелы трудно заметить, просматривая текстовые спецификации.

Таблицы решений и деревья решений — это два альтернативных приема для представления того, что система должна делать, когда в игру вступают сложные логика и решения (Beatty и Chen, 2012). В *таблице решений* (decision table) перечислены различные значения для всех факторов, влияющих на поведение системы, и приведены ожидаемые действия системы в ответ на каждую комбинацию факторов. Факторы могут быть показаны либо как утверждения с различными условиями true и false, либо как вопросы с возможными ответами «да» или «нет». Естественно, возможны таблицы решений с факторами, которые имеют более двух возможных значений.

На рис. 12-6 показана таблица решений с логикой, управляющей принятием или отклонением заказа нового химиката в системе Chemical Tracking System. На решение влияет четыре фактора:

- есть ли у пользователя, создающего запрос, право заказывать химикаты;
- имеется ли химикат в наличии на складе или у поставщика;
- включен ли химикат в список опасных химикатов, для работы с которыми необходима специальная подготовка;
- есть ли у пользователя, создающего запрос, соответствующая подготовка для работы с этим типом опасного вещества.

Номер требования					
Условие	**1**	**2**	**3**	**4**	**5**
Пользователь авторизован	false	true	true	true	true
Химикат есть в наличии	—	false	true	true	true
Химикат считается опасным	—	—	false	true	true
Сотрудник, разместивший заказ на химикат, прошел соответствующую подготовку	—	—	—	false	true
Действие					
Принять запрос			X		X
Отклонить запрос	X	X		X	

Рис. 12-6. Пример таблицы решений для Chemical Tracking System

Каждый из этих четырех факторов имеет два возможных условия, true или false. В принципе это увеличивает на 2^4, или 16, различные комбинации true/false для 16 вероятных отдельных функциональных требований. Однако на практике результатом многих комбинаций является одна и та же реакция системы. Если пользователь не имеет право заказывать химикаты, то система не примет запрос, и таким образом остальные условия несущественны (показаны прочерками в таблице решений). Таблица показывает, что результат

различных логических комбинаций — лишь пять уникальных функциональных требований.

На рис. 12-7 показано дерево решений, представляющее ту же логику. Пять прямоугольников обозначают пять возможных результатов принятия или отклонения заказа химиката. Таблицы решений и деревья решений — это хорошие способы документации требований (или бизнес-правил), позволяющие не пропустить ни одну комбинацию условий. Даже сложная таблица или дерево решений более просты для чтения, чем повторяющиеся требования в текстовом виде.

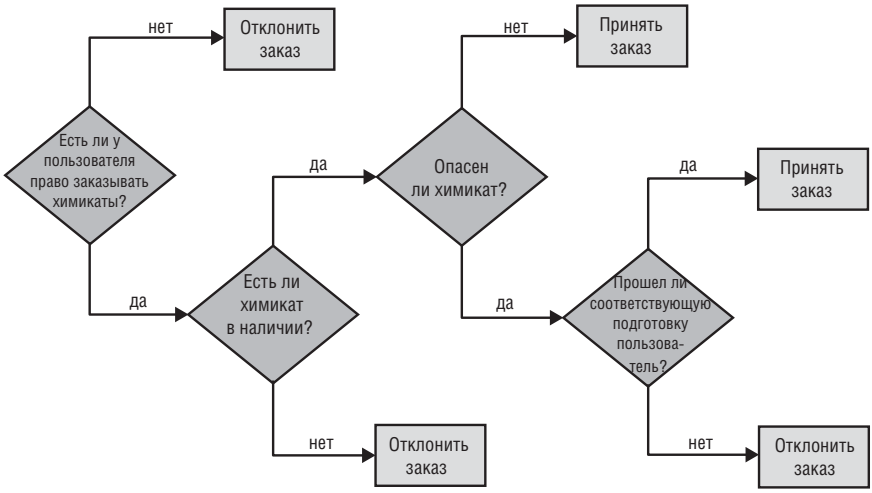


Рис. 12-7. Пример дерева решений для Chemical Tracking System

Таблицы событий и реакций

Варианты использования и пользовательские истории не всегда полезны или достаточны для обнаружения функциональности, которую должен реализовать разработчик (Wiegers, 2006). Это особенно верно в отношении систем реального времени. Представьте себе сложную транспортную развязку со многими светофорами и сигналами для пешеходов. У такой системы много вариантов использования. Водителю может требоваться проехать прямо или повернуть направо или налево. Пешеходу нужно перейти через дорогу. Еще могут быть автомобили экстренных служб, которым система должна обеспечивать «зеленую дорогу», чтобы они могли быстрее добраться к людям, нуждающимся в их помощи. На развязке могут быть камеры полиции, фотографирующие номера нарушителей, проехавших на красный сигнал. Одной этой информации разработчикам недостаточно для правильной реализации функциональности.

Есть еще один способ выявления пользовательских требований — определить внешние события, на которые система должна реагировать. *Событием* (event) называется какое-либо изменение или действие в среде пользовате-

ля, вызывающее реакцию системы ПО (Wiley, 2000). *В таблице «событие-реакция»* (event-response table) [ее также называют *таблицей событий* (event table) или *списком событий* (event list)] перечислены все такие события и ожидаемое поведение системы, которое должно последовать, как реакция на каждое событие. Существует три класса системных событий, которые показаны на рис. 12-8:

- **Бизнес-событие** Действие пользователя, инициирующее диалог с ПО, например, вызывает вариант использования (иногда это называют бизнес-событием). Последовательность событий и реакций соответствует этапам взаимодействия в данном варианте использования или диаграмме swimlane.
- **Сигнал** Это событие регистрируется, когда система получает контрольный сигнал, операцию чтения данных или прерывание, полученное от внешнего аппаратного устройства, например при изменении положения выключателя, изменении напряжения, запросе сервиса другим приложением или касании пользователем сенсорного экрана;
- **Временно́е событие** Это событие инициируется по времени, например когда на часах компьютера наступает определенное время (скажем, автоматический запуск экспорта данных в полночь) или когда истекает предварительно заданный период времени с момента определенного события (например, система фиксирует температуру, считываемую сенсором каждые 10 секунд).

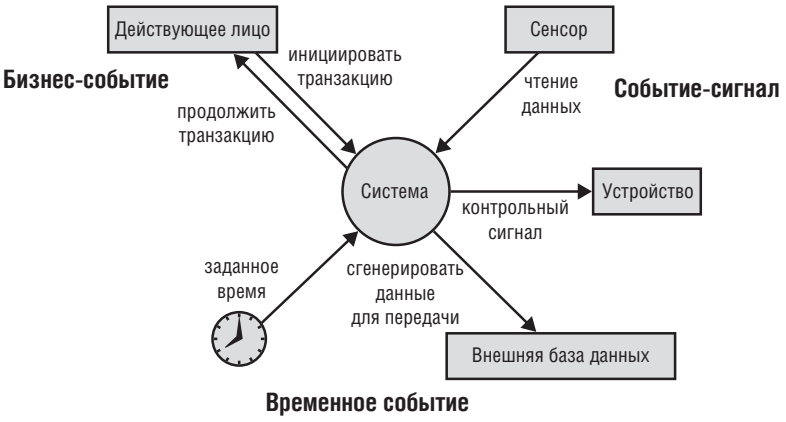


Рис. 12-8. Примеры реакции системы на бизнес-события, сигналы и временные события

Таблицы «событие — реакция» особенно хороши для управления системами реального времени. Для определения событий возьмите все состояния анализируемого объекта и определите все события, которые могут переводить объект в эти состояния. Изучите контекстные диаграммы на предмет наличия внешних сущностей, которые могут инициировать действие (инициировать событие) или требуют автоматического ответа (требуется инициирование временного события). Табл. 12-3 представляет собой простую

таблицу «событие–реакция», в которой частично описано поведение очистителей ветрового стекла автомобиля. Если не считать временного события 6, все события здесь представляют собой сигналы. Обратите внимание, что ожидаемая реакция зависит не только от события, но и от состояния, в котором находится система во время события. Например, результаты событий 4 и 5 в табл. 12-3 различаются из-за того, были ли включены «дворники» в момент, когда пользователь настроил их на периодический режим. Реакция может просто изменить некоторую внутреннюю системную информацию или привести к результату, заметному извне. В таблицу «событие-реакция» можно добавить другую информацию, в том числе:

- частоту события (сколько раз происходит событие за определенный период времени или предел количества таких событий);
- элементы данных, необходимых для обработки события;
- состояние системы после реакции на событие (Gottesdiener, 2005).

Табл. 12-3. Таблица «событие–реакция» для автомобильных стеклоочистителей

Идентификатор	Событие	Состояние системы	Реакция системы
1	Установка системы управления «дворниками» в режим медленной очистки	«Дворники»отключены, выполняют быструю очистку или периодическую очистку	Установка системы управления «дворниками» в режим медленной очистки
2	Установка системы управления «дворниками» в режим быстрой очистки	«Дворники» отключены, выполняют медленную очистку или периодическую очистку	Установка системы управления «дворниками» в режим быстрой очистки
3	Отключение системы управления «дворниками»	«Дворники» выполняют быструю очистку, медленную очистку или периодическую очистку	1. Завершение текущего цикла очистки. 2. Выключение механизма «дворников»
4	Установка системы управления «дворниками» в режим периодической очистки	«Дворники» отключены	1. Выполнение одного цикла очистки. 2. Чтение интервала очистки. 3. Инициализация таймера очистки
5	Установка системы управления «дворниками» в режим периодической очистки	«Дворники» выполняют быструю очистку или медленную очистку	1. Завершение текущего цикла очистки. 2. Чтение интервала очистки. 3. Инициализация таймера очистки

(*см. след. стр.*)

Табл. 12-3. (*окончание*)

Идентификатор	Событие	Состояние системы	Реакция системы
6	Истечение интервала таймера после завершения последнего цикла очистки	«Дворники» выполняют периодическую очистку	Выполнение одного цикла медленной очистки
7	Изменение интервала периодической очистки	«Дворники» выполняют периодическую очистку	1. Чтение интервала очистки. 2. Инициализация таймера очистки
8	Изменение интервала периодической очистки	«Дворники» отключены, выполняют быструю очистку или периодическую очистку	Нет реакции
9	Получение сигнала немедленной очистки	«Дворники» отключены	Выполнение одного цикла медленной очистки

Перечисление событий, пересекающих границу системы, — полезный метод определения рамок системы (Wiegers, 2006). Если в таблице определены и названы все возможные комбинации событий, состояний и реакций, включая условия исключений, таблица также может служить частью функциональных требований для этого фрагмента системы. Таблицу «событие-реакция» можно моделировать в таблице решений, чтобы убедиться, что проанализированы все сочетания событий и состояний системы. Однако бизнес-аналитик должен добавить дополнительные функциональные и нефункциональные требования. Например, сколько «взмахов» в минуту делают «дворники» в быстром и медленном режиме очищения? В каком режиме — быстром или медленном — выполняется периодическая очистка? Является ли периодический режим непрерывно изменяющимся или он работает дискретно? Каковы максимальный и минимальный интервал времени при работе в периодическом режиме? Если вы опустите подобную информацию, разработчику самому придется выяснять эту информацию или самому принимать решение. Помните, ваша цель — сформулировать требования настолько точно, чтобы разработчик знал, что разрабатывать, а тестировщик — как проверить правильность разработанного.

Обратите внимание, что события, перечисленные в табл. 12-3, отображают суть события, а не детали реализации. В табл. 12-3 ничего не говорится о том, как выглядит элемент управления «дворниками» или о том, как пользователь управляет ими. Дизайнер может реализовать эти требования как угодно — от традиционных элементов управления «дворниками», как в современных автомобилях, до системы распознавания голоса, реагирующей на произносимые команды: «включить дворники», «выключить дворники»,

«быстрая очистка», «очистить один раз» и т. д. Записывая пользовательские требования на уровне сути, вы избежите введения ненужных ограничений дизайна. Однако следует фиксировать все известные ограничения по дизайну, чтобы заставить разработчика думать.

Несколько слов об UML-диаграммах

Во многих проектах применяются методы объектно-ориентированного анализа, дизайна и разработки. Как правило, в предметной или рабочей области *объекты* (objects*)* соотносятся с объектами реального мира. Они представляют отдельные экземпляры, выведенные из общего шаблона, который называется *класс* (class). В описания классов входят атрибуты (данные) и действия, которые можно выполнять с этими атрибутами. *Диаграмма классов* (class diagram) — это графический способ отобразить классы, идентифицированные в ходе объектно-ориентированного анализа, и взаимоотношений между ними (см. главу 13).

Для продуктов, разработанных с помощью объектно-ориентированных методов, не требуются какие-то особые приемы разработки требований. Дело в том, что требования отражают то, что пользователям необходимо сделать с помощью системы и особенности предполагаемой функциональности, а не то, как она будет создана. Пользователям не придется вникать в объекты и классы. Однако если вы собираетесь разрабатывать систему с помощью объектно-ориентированных приемов, то анализ требований стоит начать с определения классов доменов, их атрибутов и поведения. Это упростит переход от анализа к дизайну, так как дизайнер сопоставляет объекты предметной области с системными объектами и детализирует атрибуты и операции каждого класса.

Стандартным языком объектно-ориентированного моделирования является Унифицированный язык моделирования (Unified Modeling Language, UML) (Booch, Rumbaugh и Jacobson, 1999). UML в первую очередь используется для создания моделей дизайна. На уровне абстракции, который годится для анализа требований, могут использоваться несколько моделей UML (Fowler, 2003; Podeswa, 2010):

- **Диаграммы классов**, которые отображают классы объектов предметной области: их атрибуты, поведение и свойства, а также отношения между классами. Диаграммы классов могут также применяться для моделирования данных, как показано в главе 13, но в таком ограниченном применении не полностью используются возможности диаграммы классов.
- **Диаграммы вариантов использования** — для отображения отношений между действующими лицами, являющимся внешними по отношению к системе, и вариантов использования, с которыми они взаимодействуют (см. главу 8).
- **Диаграммы действий** — отображает, как взаимодействуют различные потоки или как роли выполняют определенные действия (напри-

мер, диаграммы swimlane), или моделируют потоки бизнес-процессов. В главе 8 есть простой пример.

- **Диаграммы (или машина) состояний** — для представления различных состояний, которые могут принимать объекты системы или данных, а также разрешенных переходов между состояниями.

Моделирование в проектах гибкой разработки

Во всех проектах нужно применять модели требований для анализа этих требований с разных точек зрения, независимо от используемой методологии разработки. Выбор моделей для разных методологий может быть одинаковым. Разница между традиционным проектом и проектом гибкой разработки (agile) заключается во времени создания моделей и уровне детализации в них.

Например, в проекте гибкой разработки черновая версия уровня 0 диаграммы потоков данных делается на ранних этапах. Потом, во время итерации можно создать более подробную диаграмму потоков данных для описания границ только данной итерации. Также в проектах гибкой разработки формат моделей может быть менее постоянным и выверенным, чем в традиционном проекте. Можно создать набросок модели анализа на доске и сфотографировать ее и не хранить ее вместе с формальной документацией требований или в средстве моделирования. По мере реализации пользовательских историй модели могут обновляться (чтобы отметить уже завершенные части, можно воспользоваться цветовыми выделениями), чтобы показать, что реализуется в итерации, и увидеть дополнительные пользовательские истории, которые должны дополнить общую картину.

Ключевой момент использования моделей анализа в проектах гибкой разработки — да, в общем, и в любом другом проекте — ориентация на создание моделей, которые действительно нужны, когда нужны, и с таким уровнем детализации, которого как раз достаточно для того, чтобы заинтересованные лица проекта адекватно поняли требования. Пользовательские истории не всегда достаточны для получения нужного уровня детализации и точности, которые нужны в проекте гибкой разработки (Leffingwell, 2011). Не исключайте возможность использования каких-либо моделей просто потому, что вы работаете над проектом гибкой разработки.

Последнее напоминание

У каждого приема моделирования, описанного в этой книге, есть свои преимущества и свои ограничения. Ни одно представление не сможет охватить все стороны системы. Кроме этого, эти представления перекрываются, поэтому вам не надо создавать для своего проекта все типы диаграмм. Например, если вы создаете диаграмму «сущность–связь» и словарь данных, не стоит

рисовать диаграмму классов (или наоборот). Помните, что вы создаете модели анализа для того, чтобы обеспечить более высокий уровень понимания и взаимодействия, чем тот, что дает текстовая спецификация требований к ПО или любое другое представление требований.

Что дальше?

- Опробуйте на практике описанные в этой главе приемы моделирования, задокументировав дизайн существующей системы. Например, нарисуйте карту диалоговых окон для банкомата или веб-сайта, которые вы используете.
- В своем текущем или следующем проекте выберите прием моделирования, который дополняет текстовое описание. Сделайте набросок модели на бумаге или доске один или два раза, чтобы убедиться, что вы на верном пути, затем воспользуйтесь серийными инструментами автоматизированного проектирования ПО, которые поддерживают условные обозначения модели, которую вы используете. Попробуйте создать хотя бы одну модель, которую вы раньше не использовали.
- Попробуйте создать визуальную модель совместно с другими заинтересованными лицами. Используйте доску или наклейки для заметок для поощрения участия.
- Перечислите все внешние события, которые могут вызвать определенное поведение системы. Создайте таблицу «событие-реакция», в которой показано состояние системы в момент воздействия каждого события и реакция системы на него.

Глава 13

Определение требований к данным

Давным-давно я вел проект, в котором три программиста, иногда непреднамеренно, использовали различные имена, длину переменных и критерий проверки для одного и того же элемента. На самом деле я использовал различную длину переменной, в которой хранилось имя пользователя, в двух программах, которые написал сам. Плохие вещи происходят, когда связываешь данные различной длины. Можно переписать другие данные, получить символы-заполнители в конце или незавершенные строки символов или даже перезаписать часть кода программы, что вызовет ее отказ. Это очень нехорошо.

Мы пострадали из-за отсутствия словаря данных — общего хранилища, где определены значение, тип данных, длина, формат и допустимые значения элементов данных, используемых в приложении. Как только в команде начали более дисциплинированно определять и работать с данными, а проблемы исчезли.

Компьютерные системы работают с данными так, чтобы это приносило пользу клиентам. Хотя они и явно не показаны на трехуровневой модели требований на рис. 1-1 в главе 1, но требования к данным охватывают все три уровня. Всюду, где есть функции, есть данные. Независимо от того, как представлены данные — пиксели в видеоигре, пакеты в вызове сотового телефона, квартальные показатели вашей компании, операции банковского счета или что-либо другое, функциональность программы создается для создания, изменения, отображения, удаления, обработки и использования данных. Бизнес-аналитик должен начинать собирать определения данных по мере того, как они появляются в процессе выявления требований.

Хорошей отправной точкой являются входной и выходной потоки на контекстной диаграмме системы. Эти потоки представляют главные элементы данных на высоком уровне абстракции, которые бизнес-аналитик должен преобразовать в подробности требований в процессе их выявления. Существительные, которые пользователи упоминают в процессе выявления требований, часто указывают на важные сущности данных: заказ химиката, сотрудник, химикат, состояние, отчет об использовании химиката. В этой главе описываются приемы анализа и представления данных, которые важны для пользователей вашего приложения, а также способы определения отчетов и панелей мониторинга, которые должно создавать ваше приложение.

Моделирование отношений данных

Так же, как описанная в главе 12 диаграмма потоков данных иллюстрирует процессы, происходящие в системе, так модель данных изображает связи данных в системе. Модель данных предоставляет высокоуровневое представление данных системы, а словарь данных дает подробную картину.

Широко используется такая модель данных, как диаграмма «сущность–связь» (entity-relationship diagrams, ERD) (Robertson и Robertson, 1994). Если диаграмма «сущность–связь» представляет логические группы информации предметной области и их взаимосвязи, нужно использовать диаграмму «сущность–связь» в качестве инструмента анализа требований. Анализ диаграммы «сущность–связь» помогает понять и связать компоненты данных компании или системы, даже без предположения, что продукт будет включать базу данных. Создавая эту диаграмму в ходе разработки системы, вы определяете логическую или физическую (реализацию) структуру базы данных системы. Такое представление реализации расширяет или дополняет понимание системы, которое образовалось в процессе анализа, и оптимизирует ее реализацию, скажем, в среде реляционной базы данных.

Сущностями (entities) называются физические элементы (включая людей) или агрегации элементов данных, важных для анализируемого бизнеса или для системы, которую вы планируете создать. Сущности именуются посредством существительных в единственном числе, они показаны в прямоугольниках на диаграмме «сущность–связь». На рис. 13-1 показана часть диаграммы «сущность–связь» для Chemical Tracking System с использованием нотации Питера Чена, одной из нескольких применяемых для диаграмм такого типа систем обозначений. Обратите внимание, что объекты «Заказ химиката», «Каталог поставщика» и «Товары на складе химикатов» на диаграмме потоков данных на рис. 12-1 в главе 12 показаны в виде хранилищ данных. Другие объекты представляют действующие лица, взаимодействующие с системой (Сотрудник, разместивший заказ на химикат), физические элементы, являющиеся частью бизнес-операций (Контейнер с химикатом), и блоки данных, которые не показаны на уровне 0 диаграммы потоков данных, но показаны на ее более низком уровне (История контейнера, Химикат). В процессе проектирования физической реляционной базы данных сущности обычно становятся таблицами.

Каждая сущность описывается несколькими *атрибутами*; у разных экземпляров сущности значения атрибутов могут отличаться. Например, в атрибуты каждого химиката включены уникальный химический идентификатор, строгое название химиката и графическое представление его химической структуры. В словаре данных приводятся детальные определения этих атрибутов — это гарантирует, что объекты на диаграмме «сущность–связь» и соответствующие им хранилища данных на диаграмме потоков данных определены одинаково.

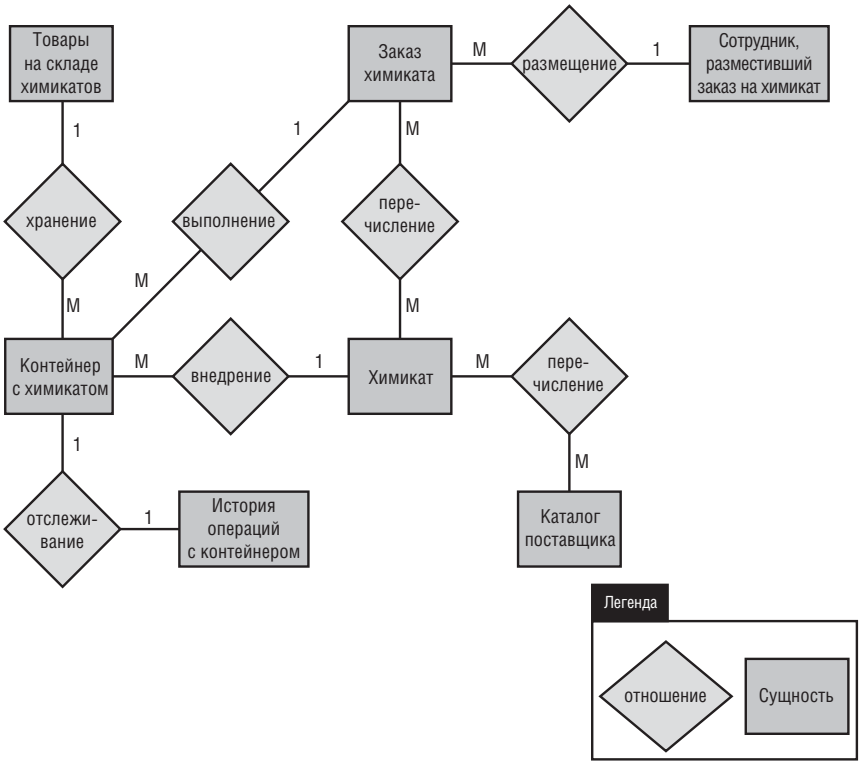


Рис. 13-1. Фрагмент диаграммы «сущность–связь» системы Chemical Tracking System

Ромбы на диаграмме «сущность–связь» обозначают *связи* (relationship), показывающие логические и числовые связи пар объектов. Названия связям даются в соответствии с характером соединений. Например, связи между элементами «Сотрудник, разместивший заказ на химикат», и «Запрос химиката» определяются как *размещение* (placing). Вы можете прочитать эту связь как «Сотрудник размещает Запрос на химикат» (справа налево, активный залог) или как «Запрос на химикат размещен Сотрудником» (слева направо, пассивный залог). Согласно некоторым правилам, фигуру в форме ромба следует назвать «размещен тем-то», что имеет смысл, только если читать диаграмму слева направо. Если перерисовать диаграмму с обратным относительным положением элементов «Сотрудник, разместивший заказ на химикат» и «Заказ химиката», тогда название связи «размещен» будет неверным при чтении слева направо: утверждение «Сотрудник размещен заказом химиката» неверно. Лучше назвать связь «размещение», а потом изменить формулировку в соответствии с грамматикой при чтении утверждения — «размещает» или «размещен».

Показывая диаграмму «сущность–связь» клиентам, попросите их проверить, все ли связи показаны корректно. Также попросите их определить все возможные сущности и связи с сущностями, которые не показаны на модели.

Мощность (cardinality) каждой связи, или ее численность, показаны цифрами или буквами на линиях, соединяющих сущности и связи. Для диаграмм «сущность–связь» используются различные нотации для обозначения мощности; нотация на рис. 13-1 используется чаще всего. Поскольку каждый сотрудник, разместивший заказ на химикат, может разместить несколько заказов, то между элементами «Сотрудник, разместивший заказ на химикат», и «Заказ химиката» существует связь «один ко многим». Это количество элементов показано цифрой *1* на линии, соединяющей элемент «Сотрудник, разместивший заказ на химикат», и связь «размещение», и буквой *M* (многие) на линии, соединяющей элемент «Запрос химиката» и связь «размещение». К другим возможным мощностям относятся: «один к одному» (каждый контейнер с химикатом отслеживается в одной истории контейнера) и «многие ко многим» (в каждом каталоге поставщика содержится множество Химикатов, а некоторые Химикаты встречаются в нескольких Каталогах поставщика). Если вам известна более точная мощность, чем просто *много* (у человека ровно два биологических родителя), вместо общего *M* можно указать конкретное число или диапазон.

В альтернативных нотациях диаграммы «сущность–связь» на линиях, соединяющих сущности и связи, для обозначения мощности используются другие символы. В нотации Джеймса Мартина, показанной на рис. 13-2, сущности по-прежнему представлены прямоугольниками, но связи между ними обозначают на соединяющей их линии. Вертикальная черта рядом с элементом «Товары на складе химикатов» указывает на мощность «1», а «гусиная лапка» рядом с элементом «Контейнер с химикатом» указывает на мощность «многие». Круг рядом с гусиной лапкой означает в элементе «Товары на складе химикатов» может находится ноль или больше элементов «Контейнер с химикатом».

Рис. 13-2. Одна альтернативная нотация в диаграмме «сущность–связь»

Помимо различных нотаций диаграммы «сущность–связь» существуют другие соглашения о моделировании данных. Команды, применяющие объектно-ориентированные методы разработки, обычно создают UML-диаграммы классов, на которых показаны атрибуты данных классов (что соответствует сущностям на диаграмме «сущность–связь»), логические связи между классами и мощность этих связей. На рис. 13-3 показана часть диаграммы классов для Chemical Tracking System. На ней показаны отношения «один ко многим» между элементами «Сотрудник, заказывающий химикат» и «Заказ химиката», каждый из которых является «классом» в прямоугольнике. Нотация «1..*» означает «один или больше». В диаграммах классов может применяться другая нотация для обозначения мощности (или множественности) (Ambler, 2005). Заметьте, что на диаграмме классов в средней

части прямоугольника также перечисляются атрибуты класса. На рис. 13-3 показана упрощенная версия нотации диаграммы классов. Когда диаграммы классов используются для объектно-ориентированного анализа и проектирования, в нижней части прямоугольника (в данном примере она пуста) обычно показываются операции, или поведения, которые объекты этого класса должны выполнять. Но для моделирования данных третий раздел прямоугольника класса оставлен пустым.

Рис. 13-3. Фрагмент UML-диаграммы классов системы Chemical Tracking System

Неважно, какая нотация используется для рисования модели данных. А *важно* то, чтобы все вовлеченные в проект люди (в идеале, все сотрудники организации), создающие такие модели, использовали одинаковые соглашения нотации и что все, кто использует или рецензирует модели, знали, как их интерпретировать.

Естественно, система должна также включать функциональность, которая делает что-то полезное с данными. Отношения между сущностями часто позволяют обнаружить такую функциональность. На рис. 13-1 показано, что между элементами «Контейнер с химикатом» и «История операций с контейнером» существует отношение «отслеживание». Поэтому нужна определенная функциональность — возможно зафиксированная в форме варианта использования, пользовательской истории или потока процессов, которая предоставляет пользователю доступ к истории операций с конкретным контейнером с химикатом. При анализе требований проекта с помощью моделей данных вы можете даже обнаружить ненужные данные, которые были обнаружены в процессе обсуждения, но потом нигде больше не использовались.

Словарь данных

Словарь данных (data dictionary) представляет собой набор подробной информации об используемых в приложении сущностях данных. Сбор информации о составе, типах данных, разрешенных значениях и т. п. в виде единого ресурса, служащего для определения критериев проверки данных, помогает разработчикам правильно писать программы и избавляет от проблем с интеграцией. Словарь данных является дополнением к словарю терминов проекта, который определяет термины предметной области или бизнес-термины приложения, сокращения и акронимы. Мы рекомендуем поддерживать словарь данных и словарь терминов отдельно.

Во время анализа требований информация словаря данных представляет элементы и структуры данных предметной области (Beatty и Chen, 2012). Эта информация попадает в дизайн в форме схем баз данных, таблиц и атрибутов, что в конечном итоге определяет имена переменных в программах. Время, потраченное на создание словаря данных, будет более чем компенсировано временем, которые вы сэкономите, избежав ошибок по причине того, что участники проекта по-разному понимают ключевые данные. Если вы регулярно обновляете словарь данных, он останется ценным средством и при обслуживании системы, и при разработке схожих систем. В противном случае он может вводить в заблуждение, предоставляя устаревшую информацию, и члены команды перестают доверять ему. Ведение словаря данных — серьезный вклад в повышение качества. Определения данных часто повторно используются в других приложениях, особенно в одном семействе продуктов. Использование единообразных определений данных в компании снижает вероятность возникновения ошибок интеграции и интерфейса. По возможности используйте существующие стандартные определения данных из хранилища корпоративных знаний, используя менее масштабный набор определений в проекте для восполнения пробелов.

По сравнению с определениями данных, разбросанными в различных местах функциональных требований, отдельный словарь данных облегчает поиск необходимой информации, а также помогает избежать ненужных повторов и несогласованности. Однажды мне пришлось рецензировать спецификации вариантов использования, определяющие элементы данных, вместе составлявших определенные структуры данных. К несчастью, эти элементы были разными в местах, где они присутствовали. Такая непоследовательность вынуждает разработчика или тестировщика выяснять, какие из определений верны и есть ли вообще верные определения. Обеспечение целостности реплицируемых структур данных по мере их изменения также непростая задача. Эти проблемы решаются путем сбора и консолидации такой информации с тем, чтобы имелся только один экземпляр каждого определения, которым могут пользоваться все заинтересованные лица.

На рис. 13-4 показана часть словаря данных для Chemical Tracking System. Используемая нотация описана чуть ниже. Упорядочивайте записи словаря данных по алфавиту, чтобы читателям было проще найти нужную информацию.

Записи словаря данных могут представлять следующие типы элементов данных:

Простейшие элементы данных Простейшим элементом данных называется тот, дальнейшая декомпозиция или упрощение которого невозможно или ненужно. К определенным на рис. 13-4 простейшим элементам данных относятся «Количество контейнеров», «Емкость», «Единица измерения», «Идентификатор заказа» и «Имя сотрудника, заказавшего химикат». Другие столбцы словаря данных можно использовать для описания относящихся к простейшему элементу данных типу, длине, диапазону числовых значений, списку разрешенных значений (как для «Единицы количества») и других уместных атрибутов.

Элемент данных	Описание	Структура или тип данных	Длина	Значения
Заказ химиката	Заказ нового химиката со склада или у поставщика	Идентификатор заказа + Сотрудник, разместивший заказ на химикат + Дата заказа + Счет затрат + 1:10{Заказанный химикат}		
Пункт назначения поставки	Место, куда нужно доставить заказанный химикат	Строение + Номер лаборатории + Отделение лаборатории		
Количество контейнеров	Количество заказываемых контейнеров определенного размера с химикатом	Положительное целое	3	
Емкость	Объем химиката в заказываемом контейнере	Целое	6	
Единица измерения	Единицы измерения, в которых указывается количество заказываемого химиката	Буквенные символы	10	Граммы, килограммы, миллиграммы и т. п.
Идентификатор заказа	Уникальный идентификатор заказа	Целое	8	Генерируемый системой порядковый номер, начиная с 1
Заказанный химикат	Описание заказываемого химиката	Идентификатор химиката + Количество контейнеров + Класс качества + Емкость + Единица измерения + (Поставщик)		

Элемент данных	Описание	Структура или тип данных	Длина	Значения
Сотрудник, разместивший заказ на химикат	Информация о сотруднике, заказывающем химикат	Имя сотрудника, заказывающего химикат + Личный номер сотрудника + Отдел + Пункт назначения поставки		
Имя сотрудника, заказывающего химикат	Имя сотрудника, заказывающего химикат	Буквенные символы	40	Может содержать пробелы, дефисы, точки и апострофы

Рис. 13-4. Фрагмент словаря данных для Chemical Tracking System

Структура Структура данных (или запись) содержит несколько элементов данных. На рис. 13-4 показаны следующие структуры данных: «Запрос химиката», «Пункт назначения поставки», «Заказанный химикат» и «Сотрудник, разместивший заказ на химикат». Столбец «Структура или тип данных» в словаре данных — место, где перечисляются элементы, из которых состоит структура, а элементы отделяются знаком «плюс» (+). Структуры могут содержать другие структуры: структура «Сотрудник, разместивший заказ на химикат» включает структуру «Пункт назначения поставки». Каждый элемент данных в структуре должен быть определен в словаре данных.

Если элемент в структуре данных необязателен (значение, которое не должно предоставляться пользователем или системой), заключите его в скобки. В структуре «Заказанный химикат» элемент данных «Поставщик» является необязательным, поскольку сотруднику, разместившему запрос, может быть безразлично или он может просто не знать, кто именно поставляет нужный химикат.

В таком макете словаря данных полезны гиперссылки (но лучше всего хранить информацию в специализированном средстве, поддерживающем такие ссылки). В качестве иллюстрации элемент «Емкость» в структуре данных «Заказанный химикат» на рис. 13-4 показан как гиперссылка. Читателю достаточно просто щелкнуть эту ссылку, чтобы перейди к определению элемента «Емкость» в другом месте словаря данных. Ссылки навигации очень полезны в объемном словаре данных, который может состоять из многих страниц или даже нескольких документов, если словарь данных проекта включает часть определений из общекорпоративного словаря данных. Хорошая мысль — включать ссылки на все элементы, находящиеся в столбце «Структура или тип данных» и определенные в словаре данных.

Повторяющаяся группа Если в структуре данных содержится несколько экземпляров элемента данных, заключите этот элемент в фигурные скоб-

ки. Покажите допустимое количество возможных повторов в формате минимум:максимум перед открывающей скобкой. Например, «Заказанный химикат» в структуре «Заказ химиката» является повторяющейся группой, которая оформляется так: *1:10{Заказанный химикат}*. Это означает, что заказ химиката должен содержать как минимум один, но не более десяти химикатов. Если максимальное количество экземпляров в повторяющемся поле неограниченно, для указания этого факта используйте «n». Например, *3:n{повтор}* означает, что определенная структура данных должна содержать не менее трех повторяющихся элементов *повтор*, а верхней границы на число повторений нет.

Точное определение элементов данных сложнее, чем может показаться. Посмотрите на такой простой тип данных, как «Буквенные символы», указанный для элемента «Имя сотрудника, заказывающего химикат» на рис. 13-4. Чувствительно ли имя к регистру, то есть «Карл» и «карл» — это одно или разные имена? Должна ли система приводить весь текст к верхнему или нижнему регистру или сохранять регистр значений, которые искали, или введенных пользователем значений или отбрасывать строки с регистрами букв, отличающимися от ожидаемых? Могут ли использоваться другие символы помимо 26 букв английского алфавита, такие как пробелы, дефисы, точки или апострофы — все они могут присутствовать в именах? Разрешен только английский алфавит или также буквы алфавитов с диакритическими знаками, такими как тильда (~), умляут (¨), ударения (` или ´), седиль (¸) и другие? Такие точные определения очень важны для разработчика, чтобы точно понимать, как проверять вводимые данные. Еще одна возможность вариаций — формат отображения элементов данных. Например, существует масса вариантов представления даты и времени, отличающихся в разных странах. Стивен Уитхолл (Stephen Withall, 2007) описывает много нюансов, о которых нужно помнить при определении различных типов данных.

Анализ данных

При выполнении анализа данных нужно сопоставлять различные представления информации между собой для обнаружения пробелов, ошибок и противоречий. Сущности в диаграмме «сущность–связь» скорее всего определены в словаре данных. Потоки данных и хранилища в диаграмме потоков данных скорее всего есть где-то в диаграмме «сущность–связь», а также в словаре данных. Отображаемые поля в спецификации отчета также должны присутствовать в словаре данных. Во время анализа данных можно сравнить эти дополняющие друг друга представления, чтобы найти ошибки и дополнительно уточнить свои требования к данным.

Один из точных способов поиска недостающих требований — матрица CRUD (Create, Read, Update, Delete — создание, чтение, обновление, удаление). Она позволяет соотнести действия системы с элементами данных (отдельными или их совокупностями), что дает представление о том, где и как

каждый элемент данных создается, считывается, обновляется и удаляется. (Некоторые добавляют к названию матрицы букву *L*, указывая, что элемент данных является списком (list), или *M* или *C* — соответственно для обозначения перемещения или копирования данных из одного места в другое. Для простоты мы будем придерживаться обозначения CRUD.) В зависимости от используемых способов анализа требований можно исследовать различные типы соответствий, в том числе:

- элементы данных и системные события (Ferdinandi, 2002; Robertson и Robertson, 2013);
- элементы данных и задачи пользователей или варианты использования (Lauesen, 2002);
- классы объектов и варианты использования (Armour и Miller, 2001).

На рис. 13-5 показана матрица CRUD для части системы Chemical Tracking System. Каждая ячейка указывает, как вариант использования, определенный в крайнем левом столбце, использует элементы данных, показанные в остальных столбцах. Вариант использования может создать (C), прочитать (R), обновить (U) или удалить (D) сущность. После создания матрицы посмотрите, нет ли ячейки столбца, в которой нет одной из этих букв. Например, если сущность обновлена, но до этого ее не создавали, то откуда она взялась?

Сущность / Вариант использования	Заказ	Химикат	Сотрудник, разместивший заказ на химикат	Каталог поставщика
Разместить заказ	C	R	R	R
Изменить заказ	U, D		R	R
Управлять складом химикатов		C, U, D		
Создать отчет по заказу	R	R	R	
Редактировать сотрудников, имеющих право размещать заказы			C, U	

Рис. 13-5. Пример матрицы CRUDL для Chemical Tracking System

Обратите внимание, что ни одна ячейка в столбце «Сотрудник, разместивший заказ на химикат» не содержит *D*. То есть ни в одном из случаев использования на рис. 13-5 нельзя удалить сотрудника из списка людей, заказывавших химикаты. Интерпретировать это можно тремя способами:

- удаление сотрудника, разместившего заказ на химикат, не является ожидаемой функцией Chemical Tracking System;
- мы пропустили вариант использования, который удаляет сотрудника, разместившего заказ на химикат;
- вариант использования «Редактировать сотрудников, имеющих право размещать заказы» некорректный. Предполагается, что пользователь мо-

жет удалить из списка сотрудника, размещающего заказ на химикат, но в настоящее время эта функциональность не указана в вариантах использования.

Мы не знаем, какая интерпретация правильна, но CRUD — это надежный способ для обнаружения недостающих требований.

Спецификация отчетов

Многие приложения генерируют отчеты на основе данных в одной или большем числе БД, в файлах и в других источниках информации. Отчеты могут представлять собой традиционные таблицы со строками и столбцами данных, самые разнообразные диаграммы и графики или их сочетание. Изучение содержимого и формата необходимых отчетов — важна часть разработки требований. Спецификация отчетов охватывает требования (какая информация идет в отчет и как она организуется) и дизайн (как отчет должен выглядеть). В этом разделе даются советы о том, о каких характеристиках отчетов надо спрашивать и какую информацию записывать. Также дан шаблон для определения отчетов.

Сбор требований к отчетности

Если вы бизнес-аналитик, работающий с клиентами над определением требований к отчетности информационной системы, вам стоит задать следующие вопросы:

- Какие отчеты вы используете в настоящее время? (Некоторые отчеты в существующей системе и вручную создаваемые отчеты может потребоваться перенести в новую систему.)
- Какие из существующих отчетов потребуется изменить? (Проект новой или обновленной информационной системы дает возможность обновить отчеты, которые не совсем отвечают текущим потребностям.)
- Какие отчеты сейчас создаются, но не используются? (Возможно, эти отчеты не нужно переносить в новую систему.)
- Не могли бы вы описать какие-либо стандарты — отдела, компании или общегосударственные, которым должны отвечать отчеты, например единообразие вида и стиля или соответствие нормативным документам? (Получите копии этих стандартов и примеры существующих отчетов, которые им соответствуют.)

Уитхолл (Withall, 2007) описывает схему и шаблон для определения требований к отчету. Джой Битти и Энтони Чен (Joy Beatty и Anthony Chen, 2012) также предоставляют обширные советы по спецификации отчетов. Далее приведены некоторые вопросы для изучения каждого запрошенного клиентом отчета. Первый набор вопросов относится к контексту и использованию отчета:

- Как называется отчет?
- Какова задачи или бизнес-цель отчета? Как адресаты отчета используют содержащуюся в нем информацию? Какие решения принимаются на основе отчета и кем?
- Создается ли отчет вручную? Если да, то как часто и каким классом пользователей?
- Создается ли отчет автоматически? Если да, то как часто и какое событие служит инициатором создания отчета?
- Каковы типичный и максимальный размеры отчета?
- Есть ли необходимость в создании панели мониторинга (dashboard), на которой бы отображались несколько отчетов и/или графиков? Если да, то должна ли у пользователя быть возможность развертки или свертки каких-либо элементов панели мониторинга?
- Как используется отчет после его создания? Он отображается на экране, отправляется получателю, экспортируется в электронную таблицу или автоматически печатается? Хранится ли где-то в архиве для использования в будущем?
- Есть ли какие-либо ограничения в плане безопасности, приватности или управления, которые дают доступ к отчету только вполне определенным пользователям или классам пользователей или которые ограничивают объем данных в отчете в зависимости от того, кто его создает? Укажите все соответствующие бизнес-правила, относящиеся к безопасности.

Следующие вопросы служат для выявления информации о самом отчете:

- Каковы источники данных и критерии выборки информации из хранилища?
- Какие параметры доступны пользователю для выбора?
- Какие требуются вычисления и другие преобразования данных?
- Каковы критерии сортировки, разбиения на страницы и определения итогов?
- Как должна реагировать система, если при попытке построения отчета запрос не возвращает никаких данных?
- Должны ли лежащие в основе отчета данные быть доступными пользователю для получения нерегламентированной отчетности?
- Может ли данный отчет служить шаблоном для набора аналогичных отчетов?

Особенности определения отчетов

Следующие советы будут полезны бизнес-аналитику при выявлении требований к отчетности.

Предлагайте другие варианты Когда пользователь запрашивает создание определенного отчета, бизнес-аналитик может предложить варианты из-

менения или улучшения отчета, что может повысить его бизнес-ценность. Один из вариантов — просто иначе упорядочить данные, например путем предоставления возможности упорядочивать элементы данных по какому-то параметру, который пользователь изначально не запрашивал. Можно дать пользователю возможность менять порядок столбцов. Еще один вариант — свертка или развертка информации. Сводный отчет агрегирует подробные результаты в сжатое представление более высокого уровня. Развертка, или детализация (drill down), означает получение отчета, в котором отображается детализация сводных данных.

Поиск данных Обеспечьте наличие в системе данных, необходимых для наполнения отчета. Пользователи мыслят в терминах получения нужных им результатов, что подразумевает определенные входные данные и источники, которые предоставляют нужные данные. Этот анализ может обнаружить ранее неизвестные требования по доступу и генерированию необходимых данных. Определите все бизнес-правила, которые должны применяться для вычисления выходных данных.

Рассчитывайте на рост в будущем Пользователи могут запрашивать определенные отчеты, основываясь на их начальных представлениях, сколько данных или сколько параметров могут быть задействованы. По мере роста системы со временем начальный макет отчета, который хорошо работал с небольшими объемами данных, может оказаться негодным. Например, столбчатое представление определенного числа подразделений компаний может умещаться на одной странице. Но при увеличении числа подразделений вдвое могут возникать неприглядные разбиения страниц или необходимость горизонтальной прокрутки отображаемого отчета. Может потребоваться изменить макет с портретного в альбомный режим или транспонировать информацию из представления в виде столбцов в вид со строками.

Следите за аналогиями Несколько пользователей, или даже один и тот же пользователь, могут запрашивать похожие, но не идентичные отчеты. Посмотрите, нельзя ли объединить эти варианты в один отчет, который достаточно гибок, чтобы удовлетворить разные потребности без излишних усилий на разработку или обслуживание. Иногда вариативность можно обеспечить с помощью параметров.

Различайте статические и динамические отчеты Статические отчеты отображают на бумаге или экране компьютера данные по состоянию на определенный момент времени. Динамические отчеты представляют интерактивное представление данных в реальном времени. При изменении лежащих в основе отчета данных система автоматически обновляет вид отчета. Такая функциональность есть в моей бухгалтерской программе. Если при просмотре отчета о расходах я введу новый недавно выписанный мной чек, отчет немедленно обновится с учетом новой информации. Укажите, какой тип отчета требуется и соответствующим образом корректируйте требования.

Создавайте прототипы отчетов Часто бывает полезной модель отчета, которая иллюстрирует возможный подход и стимулирует пользователей на

предоставление отзывов, или для иллюстрации требуемого макета можно использовать похожий уже существующий отчет. При использовании такого прототипа во время обсуждения требований участники могут формулировать ограничения дизайна, что может быть, а может и не быть полезным. В других случаях, разработчик может создать пример макета отчета и попросить клиента сообщить свое мнение о нем. При создании моделей отчетов используйте разумные данные, чтобы отчет выглядел реалистично для пользователей, которые будут его оценивать.

Шаблон спецификации отчета

На рис. 13-6 показан шаблон для создания спецификации отчета. Часть этих элементов отчета будут определены во время выявления требований, остальные определяются во время дизайна. Требования могут определять содержимое отчета, а в процессе дизайна определяются конкретный макет и форматирование. Наличие существующих стандартов отчетов может помочь сразу определить некоторые элементы шаблона.

Не все эти элементы и вопросы применимы ко всем отчетам. Также возможны варианты размещения этих элементов. Заголовок отчета может отображаться только в начале первой страницы или на каждой странице отчета. Используйте информацию на рис. 13-6 как руководство для бизнес-аналитиков, клиентов, разработчиков и тестировщиков для понимания требований и ограничений дизайна конкретного отчета.

Элемент отчета	Описание
Идентификатор отчета	Номер, код или метка, служащие для идентификации или классификации отчета
Заголовок отчета	Название отчета Позиция заголовка на странице Нужно ли включать параметр, использованный при построении отчета (например, диапазоны данных)?
Цель отчета	Краткое описание проекта, общая информация, контекст или бизнес-цель, для которой создается отчет
Решения, принимаемы на основе отчета	Бизнес-решения, которые принимаются на основе информации отчета
Приоритет	Относительный приоритет реализации этой функциональности отчета
Пользователи отчета	Классы пользователей, которые будут создавать отчет или применять его для принятия решений
Источники данных	Приложения, файлы, базы или хранилища данных, из которых будут извлекаться данные

(*см. след. стр.*)

Элемент отчета	Описание
Частота и использование	Каков тип отчета: статический или динамический?
	Как часто будет генерироваться отчет: раз в неделю, в месяц или по требованию?
	Какой объем данных используется или сколько транзакций обрабатывается при построении отчета?
	Какие условия или события инициируют построение отчета?
	Создается ли отчет автоматически? Требуется ли ручное вмешательство?
	Кто получатель отчета? Как отчет предоставляется получателю (отображается в приложении, отправляется по электронной почте, печатается, просматривается на мобильном устройстве)?
Время доступа	Как быстро отчет должен предоставляться пользователям после его запроса?
	Насколько свежими и актуальными должны быть данные при построении отчета?
Визуальный макет	Альбомное или портретное представление
	Размер бумаги (или тип принтера), используемый для создания бумажных отчетов
	Если отчет включает графики, определите типы всех графиков, их внешний вид и параметры: заголовки, масштаб и метки осей, источники данных и т. п.
Верхний и нижний колонтитулы	Ниже перечислены элементы, которые могут располагаться в верхнем или нижнем колонтитуле отчета. Для каждого включаемого элемента нужно указать его место на странице и внешний вид, в том числе тип, размер и начертание шрифта, цвет, регистр и выравнивание текста. Что делать, когда заголовок или другой текст не вмещается в отведенное место: обрезать, перенести на следующую строку или сделать что-то другое?
	Заголовок отчета
	Нумерация и формат страниц (например, «Страница x» или «Страница x из y»)
	Примечания к отчету (например, «Из этого отчета исключены сотрудники, отработавшие в компании менее одного месяца»)
	Метка времени создания отчета
	Имя человека, инициировавшего построение отчета
	Источник(и) данных, в частности приложение хранилища данных, которое консолидирует данные из многих источников
	Начальная и конечная даты отчета
	Идентификационная информация организации (название компании, отдел, логотип и другие графические элементы)
	Заявление о неразглашении или уведомление об авторских правах

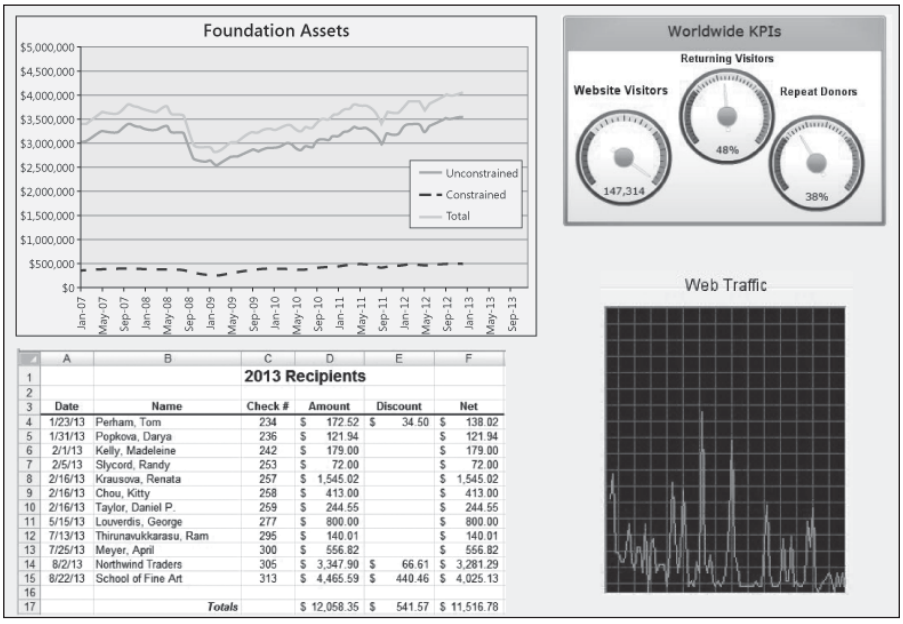
Элемент отчета	Описание
Тело отчета	Критерии отбора данных для отчета (логика определения, какие данные должны и какие не должны войти в отчет)
	Включаемые в отчет поля
	Определяемый пользователем текст или параметры для модификации меток полей
	Заголовки и форматы столбцов и строк: текст, шрифт, размер, цвет, начертание, регистр, выравнивание
	Разметка полей данных в столбцах и строках или позиционирование и параметры графиков и диаграмм
	Формат отображения каждого поля: шрифт, размер, цвет, выделение, регистр, выравнивание, округление чисел, число десятичных знаков и форматирование, специальные символы ($, %, запятые, десятичные точки, ведущие или завершающие символы-заполнители
	Как обрабатывать переполнения числовых и текстовых полей
	Вычисления и другие преобразования данных, выполняемые для получения отображаемых данных
	Критерии сортировки в каждом поле
	Критерии фильтрации или параметры, используемые для ограничения запроса, выполняемого перед построением отчета
	Группировка и промежуточные итоги, включая форматирование итогов и строки перебивки промежуточных итогов
	Критерии разбиения на страницы
Признак конца отчета	Внешний вид и позиция признака, обозначающего конец отчета
Интерактивность	Если отчет является динамическим или строится в интерактивном режиме, какие возможности изменения содержания или внешнего вида отчета должны быть доступными пользователю (свертка или развертка частей, ссылки на другие отчеты, развертка до источников данных)?
	Должны ли параметры отчетов сохраняться между сеансами построения отчета?
Ограничения безопасности доступа	Любые ограничения доступа конкретных людей, групп или организаций к построению или просмотру отчета или объема данных, которые разрешено выбирать для включения в отчет

Рис. 13-6. Шаблон спецификации отчета

Панели мониторинга

Панель мониторинга (dashboard) — изображение на экране или в печатном документе с множественными текстовыми и/или графическими представле-

ниями данных, предоставляющее консолидированное многомерное представление происходящего в организации или процессе. В компаниях панели мониторинга часто используют для сведения в одно место информации по продажам, расходам, ключевых показателей эффективности (KPI) и т. п. Приложения для торговли на бирже содержат совершенно дикое (для новичка) разнообразие графиков и данных, которые опытный глаз брокера считывает практически моментально. Некоторые панели мониторинга могут динамически обновляться в реальном времени по мере изменения данных. На рис. 13-7 показана гипотетическая панель мониторинга благотворительного фонда.



	A	B	C	D	E	F
1			2013 Recipients			
2						
3	Date	Name	Check #	Amount	Discount	Net
4	1/23/13	Perham, Tom	234	$ 172.52	$ 34.50	$ 138.02
5	1/31/13	Popkova, Darya	236	$ 121.94		$ 121.94
6	2/1/13	Kelly, Madeleine	242	$ 179.00		$ 179.00
7	2/5/13	Slycord, Randy	253	$ 72.00		$ 72.00
8	2/16/13	Krausova, Renata	257	$ 1,545.02		$ 1,545.02
9	2/16/13	Chou, Kitty	258	$ 413.00		$ 413.00
10	2/16/13	Taylor, Daniel P.	259	$ 244.55		$ 244.55
11	5/15/13	Louverdis, George	277	$ 800.00		$ 800.00
12	7/13/13	Thirunavukkarasu, Ram	295	$ 140.01		$ 140.01
13	7/25/13	Meyer, April	300	$ 556.82		$ 556.82
14	8/2/13	Northwind Traders	305	$ 3,347.90	$ 66.61	$ 3,281.29
15	8/22/13	School of Fine Art	313	$ 4,465.59	$ 440.46	$ 4,025.13
16						
17		*Totals*		$ 12,058.35	$ 541.57	$ 11,516.78

Рис. 13-7. Гипотетическая панель мониторинга благотворительного фонда

При сборе требований для панели мониторинга применяется следующая последовательность действий по выявлению и анализу требований. Многие из этих шагов также полезны при определении отдельных отчетов, как описано ранее в этой главе.

- Определите, какая информация нужна пользователям панели мониторинга для принятия решений или выполнения выбора. Понимание того, как будут использоваться представляемые данные, помогает выбрать наиболее подходящий способ отображения.
- Определите источники всех представленных данных, чтобы можно было обеспечить доступ приложения ко всем источникам, а также узнать их тип — статические или динамические.
- Выберите наиболее подходящий тип отображения каждого набора связанных данных. Должны ли они отображаться как: простая таблица с данными, изменяемая электронная таблица, содержащая формулы, блок текста,

гистограмма, круговая диаграмма, линейчатый график, видео или какой-либо другой вид диаграммы или графика?

- Определите оптимальный макет и относительные размеры различных элементов панели мониторинга, основываясь на понимании того, как пользователь будет воспринимать и применять информацию.
- Определите детали каждого элемента панели мониторинга. То есть каждый из их надо рассматривать как отдельный мини-отчет. Для этого пригодятся вопросы, перечисленные в разделе «Сбор требований к отчетности» ранее в этой главе, и шаблон на рис. 13-6. Далее перечислены вопросы, которые может потребоваться изучить:
  - Если отображаемые данные динамические, то как часто они должны обновляться или расширяться и каким образом? Например, должны ли текущие данные в окне фиксированного размера смещаться за экран влево, по мере добавления новой информации слева?
  - Какие параметры должны быть доступны пользователю для изменения, например диапазон дат?
  - Требуется ли пользователю условное форматирование, чтобы разделы элементов изменялись в зависимости от данных? Это полезно при создании отчетов о состоянии или ходе работы: зеленый свет может означать, что данные отвечают условиям, то есть «Хорошо!», желтый может означать «Внимание!», а красный — «Все очень плохо!» При использовании цветов не забывайте использовать другие варианты выделений, чтобы элементом панели мониторинга могли пользоваться дальтоники и те, кто использует монохромное отображение.
  - На каких экранах нужны горизонтальные или вертикальные полосы прокрутки?
  - Будет ли у пользователя возможность увеличить элемент панели мониторинга, чтобы увидеть больше подробностей? Должна ли у пользователя быть возможность минимизировать или закрывать элементы, чтобы освободить место на экране? Должны ли заданные пользователем параметры и изменения сохраняться между сеансами использования панели мониторинга?
  - Потребуется ли пользователю изменить форму элемента, например переключаться между табличным и графическим представлением?
  - Может ли пользователю потребоваться развернуть какой-то из элементов, чтобы увидеть более детальный отчет или лежащие в его основе данные?

Создание прототипа панели мониторинга — отличный способ взаимодействия с заинтересованными лицами для обеспечения того, чтобы макет и стили представления отвечали их потребностям. Можно набросать возможные компоненты на наклейках для заметок и попросить заинтересованных лиц упорядочить их так, как им нравится. Итеративный подход — ключ как к уточнению требований, так и поиску альтернативных вариантов дизайна.

Как обычно бывает со спецификацией требований, уровень подробностей, который нужно обеспечить при спецификации отчетов или панелей мониторинга, зависит от того, кто принимает решение, как они должны выглядеть, и как принимаются такие решения. Чем больше деталей вы хотите делегировать дизайнеру, тем меньше информации нужно предоставлять в требованиях. И, как всегда, тесное сотрудничество бизнес-аналитика, представителя пользователей и разработчиков способствует тому, что все будут довольны результатом.

Что дальше?

- Возьмите объект данных средней сложности из одного из ваших приложений и определите его с помощью нотации словаря данных, представленной в этой главе.
- Создайте диаграмму «сущность–связь» для части объектов данных своего приложения. Если у вас нет инструмента моделирования данных, для начала воспользуйтесь таким средством, как Microsoft Visio.
- Для практики определите один из существующих отчетов вашего приложения в соответствии с приведенным на рис. 13-6 шаблоном. Скорректируйте шаблон по мере необходимости в соответствии с характером отчетов, создаваемых в вашем приложении.

Глава 14

Обратная сторона функциональности: атрибуты качества ПО

«Привет, Сэм, это Кларис. Сегодня я провожу занятие в новом классе, но система кондиционирования и отопления работает ужасно громко. Мне по сути приходится перекрикивать вентилятор, и я практически уже сорвала голос. Вы ведь отвечаете за обслуживание. Почему система работает так громко? Может, она сломана?»

«Все работает нормально, — ответил Сэм. — Система в этом классе соответствует требованиям, которые дали мне инженеры. Она заменяет правильный объем воздуха в минуту, поддерживает заданную температуру в диапазоне от 16 до 30 градусов и обладает всеми запрошенными возможностями программирования. Никто ничего не говорил о шуме, поэтому я приобрел самую дешевую систему, отвечающую заданным требованиям».

Кларенс сказала: «С поддержанием температуры все нормально. Но ведь это класс! Слушатели практически не слышат меня. Нам придется установить звукоусиливающую аппаратуру или обзавестись менее шумной системой обогрева и кондиционирования. Что ты предлагаешь?»

Сэм оказался в замешательстве. «Кларис, система отвечает всем требованиям, которые мне дали, — снова повторил он. — Если бы я знал, что уровень шума так важен, я бы купил другую систему, но теперь замена будет стоить немалых денег. Может тебе попробовать мятные леденцы, чтобы совсем не потерять голос?»

Успех программы определяется не только наличием всей нужной функциональности. У пользователей также есть ожидания, часто невысказанные, о том, *насколько хорошо* должен работать продукт. К таким ожиданиям относится то, как легко его использовать, как быстро он работает, как редко отказывает, как ведет себя в неожиданных ситуациях и — возможно — как громко шумит. В целом эти характеристики называют *атрибутами качества ПО*, *факторами качества*, *требованиями к качеству* или *требованиями по уровню сервиса* и считают основной частью нефункциональных требований. Другими классам нефункциональных требований являются ограничения (они обсуждаются в конце этой главы) и требования к внешним интерфейсам (см. главу 10).

Подробнее о термине «нефункциональные требования» см. врезку «Если они нефункциональные, то что они из себя представляют?» в главе 1.

Иногда люди слишком увлекаются дебатами насчет того, к какому классу отнести ту или иную потребность — к функциональным или нефункциональным требованиям. Здесь важнее определить само требование, а категория не так важна. Эта глава позволит обнаружить и определить нефункциональные требования, которые не удалось обнаружить другими методами.

Часто именно атрибуты качества отличают продукт, который просто работает так, как ожидалось, от продукта, который вызывает у клиентов восхищение. Отличные программные продукты — результат оптимального баланса противоречивых характеристик качества. Если в ходе сбора информации о требованиях вы досконально не выясните ожидания клиента, относящиеся к качеству, то вам крупно повезет, если продукт их удовлетворит. Но, как правило, более частый исход — разочарованные пользователи и расстроенные разработчики.

Атрибуты качества служат источником многих функциональных требований. Они также определяют важные решения по архитектуре и дизайну. Гораздо дороже перестраивать готовую систему, чем спланировать необходимые цели по качеству с самого начала. Вспомните о многочисленных обновлениях безопасности, которые периодически выпускают поставщики операционных систем и популярных приложений. Небольшие дополнительные усилия по обеспечению безопасности во время разработки могут предотвратить значительные затраты и избавить пользователей от неудобств.

Так дело не пойдет!

Атрибуты безопасности могут стать причиной успеха или провала вашего продукта. Одна крупная компания потратила миллионы долларов на замену приложения центра обработки вызовов на старой системе с зелеными монохромными экранами на модную версию на Windows. После всех затрат по замене сотрудники центра обработки вызовов отказались использовать новую систему, потому что в ней была очень сложная навигация. Опытные пользователи в одночасье оказались без привычных «быстрых» сочетаний клавиш, которые позволяли эффективно работать в старой системе. Теперь для перемещения в приложении им приходилось использовать мышь, и работа сильно замедлилась. Поначалу руководство выбрало жесткую линию поведения: «Мы в приказном порядке заставим их использовать новую систему». Но сотрудники центра упорствовали. Что оставалось делать? Эти люди принимают клиентские заказы, поэтому если сотрудники не будут использовать новую систему, компания физически не сможет отключить старую систему, потому что рискует потерять все эти заказы. Пользователи просто не хотят снижать свою производительность из-за «новой и более совершенной» системы. Команде разработчиков пришлось перепроектировать пользовательский интерфейс и добавить старые сочетания клавиш — только после этого пользователи приняли новую программу, но это на месяцы задержало ее выпуск.

Атрибуты качества ПО

К атрибутам качества можно отнести десятки характеристик продукта, хотя для большинства проектов хватает всего нескольких. Если разработчикам известно, какие характеристики наиболее важны для успеха проекта, они могут выбрать соответствующие приемы работы над архитектурой и дизайном, которые позволят достичь определенного качества. Для классификации атрибутов качества применяются различные схемы (DeGrace и Stahl, 1993; IEEE, 1998; ISO/IEC, 2007; Miller, 2009; ISO/IEC, 2011). Некоторые авторы разработали обширные иерархии, которые группируют связанные атрибуты на несколько основных категорий.

Один из способов классификации атрибутов качества основан на разделении характеристик, которые проявляются в период выполнения (внешнее качество), и тех, что не проявляются (внутреннее качество) (Bass, Clements и Kazman, 1998). Внешние характеристики главным образом важны для пользователей, а внутренние имеют значение для разработчиков и службы технической поддержки. Внутреннее качество косвенно влияет на мнение клиента за счет упрощения возможных улучшений продукта, его корректировки, тестирования и миграции на другие платформы.

В табл. 12-1 перечислено несколько атрибутов качества обеих категорий, которые необходимо принимать во внимание в любом проекте. Потребность в конкретных атрибутах зависит от типа проекта:

- встроенные системы: производительность, эффективность, надежность, устойчивость, безопасность, эргономика (см. главу 26);
- интернет-приложения и корпоративные приложения: доступность, целостность, функциональная совместимость, производительность, масштабируемость, безопасность и удобство использования;
- настольные и мобильные системы: производительность, безопасность, удобство и простота использования.

Для различных частей системы необходимы различные сочетания атрибутов качества. Для одних крайне важна производительность, а для других — практичность. В вашей среде могут быть другие уникальные атрибуты качества, о которых здесь ничего не говориться. Например, в компании, занимающейся играми, могут требовать особой эмоциональной атмосферы в разрабатываемом ПО (Callele, Neufeld и Schneider, 2008).

Атрибутам качества посвящен раздел 6 шаблона спецификации требований к ПО в главе 10. Если какие-то из требований к качеству относятся к определенным функциям, компонентам, функциональным требованиям или пользовательским историям, их нужно связать с соответствующими элементами в хранилище требований.

Табл. 14-1. Некоторые атрибуты качества ПО

Внешнее качество	Краткое описание
Доступность	Насколько сервисы системы доступны, когда и где они нужны
Удобство установки	Насколько просто правильно установить, удалить и повторно установить приложение
Целостность	Насколько хорошо система защищает от неточности и потери данных
Совместимость	Насколько просто система может взаимодействовать и обмениваться данными с другими системами и компонентами
Производительность	Как быстро и предсказуемо система реагирует на ввод информации пользователем и на другие события
Надежность	Как долго система работает до первого сбоя
Устойчивость	Как хорошо система реагирует на неожиданные условия работы
Защита	Насколько хорошо система защищает от повреждений
Безопасность	Как хорошо система защищает от неправомерного доступа к приложению и его данным
Удобство использования	Как просто людям научиться использовать систему
Внутреннее качество	**Краткое описание**
Эффективность	Насколько эффективно система использует ресурсы компьютера
Возможность модификации	Насколько легко обслуживать, модифицировать, улучшать и реструктурировать систему
Переносимость	Насколько легко заставить систему работать в другой операционной среде
Возможность повторного использования	В какой степени компоненты могут использоваться в других системах
Масштабируемость	Как хорошо система справляется с увеличением числа пользователей, транзакций, серверов и других расширений
Проверяемость и тестируемость	Как быстро разработчики и тестировщики могут подтвердить, что система реализована правильно

Изучение атрибутов качества

В идеальном мире у каждой системы максимально возможные значения всех этих атрибутов. Она постоянно доступна, никогда не сбоит, моментально предоставляет результаты, которые всегда правильны, отражает все попытки неправомерного доступа и никогда не вводит пользователя в заблуждение. В реальности существуют компромиссы и противоречия между отдельными атрибутами, которые не позволяют достичь максимума всех этих атрибутов

одновременно. Поскольку все перечисленное невозможно, советую вам выяснить с помощью табл. 14-1, какие атрибуты наиболее важны для успеха вашего проекта. Затем определите конкретные цели по качеству, основываясь на жизненно важных атрибутах, чтобы дизайнеры смогли принять соответствующие решения.

В различных проектах необходимы различные сочетания атрибутов качества. Джим Броссо (Jim Brosseau, 2010) рекомендует следующий практический подход к выявлению и определению самых важных атрибутов в проекте. Он предоставляет применяемую для анализа электронную таблицу по адресу: *www.clarrus.com/resources/articles/software-quality-attributes.*

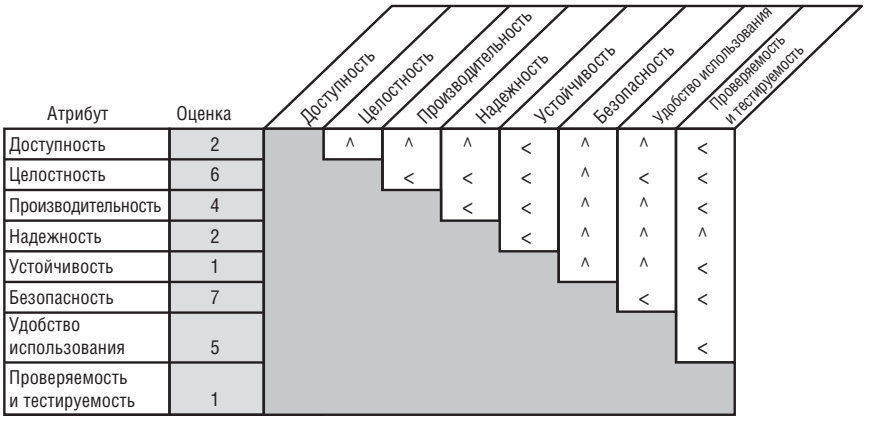
Этап 1. Использование вначале широкого набора атрибутов качества Начните с обширного набора атрибутов качества, например с тех, что перечислены в табл. 14-1. Такой широкий начальный набор снижает вероятность пропустить важную сторону качества.

Этап 2. Сокращение списка Проанализируйте круг заинтересованных лиц, чтобы оценить, какие атрибуты могут быть важными для проекта. (Обширный список возможных заинтересованных лиц проекта см. на рис. 2-2 в главе 2.) Для терминала регистрации в аэропорту может особо требоваться удобство использования (потому что пользователи будут работать с ним нечасто) и безопасность (потому что он обрабатывает платежи). Неприменимые к проекту атрибуты не требуются в дальнейшем рассмотрении. Зафиксируйте в письменном виде обоснования того, почему нужно или не нужно учитывать тот или иной атрибут качества.

Но надо понимать, что если вы не укажете цели по качеству, не стоит удивляться, что продукт не будет обладать ожидаемыми характеристиками. Поэтому важно собрать информацию у многих заинтересованных лиц. На практике обычно есть атрибуты, которые явно находятся в границах проекта и которые совершенно неважны для проекта. Но есть небольшая группа, применимость которой к проекту требуется в дополнительном обсуждении.

Этап 3. Определение приоритетов атрибутов Определение приоритетов применимых атрибутов задает направление будущих обсуждений по выявлению требований. Попарное сравнение для ранжирования может эффективно работать в небольших списках, как этот. Рис. 14-1 иллюстрирует, как применить электронную таблицу Броссо для оценки атрибутов качества для терминала регистрации в аэропорту. При заполнении каждой ячейки на пересечении двух атрибутов попытайтесь ответить на следующий вопрос: «Если бы были только эти два атрибута, то какому из них отдать предпочтение?» Знак «меньше» (<) в ячейке означает, что атрибут в строке важнее, а знак вставки «крышечка» (^), — что важнее атрибут в столбце. Например, при сравнении доступности и целостности я решил, что последняя важнее. Если терминал не работает, пассажир всегда может зарегистрироваться у сотрудника на стойке регистрации (хотя ему может потребоваться отстоять очередь). Но если терминал будет предоставлять неправильную информацию, пассажир будет *очень* недоволен. Поэтому в ячейке пересечения доступности и целостности

я поставил знак вставки, чтобы указать, что целостность важнее.



Атрибут	Оценка	Доступность	Целостность	Производительность	Надежность	Устойчивость	Безопасность	Удобство использования	Проверяемость и тестируемость
Доступность	2		^	^	^	<	^	^	<
Целостность	6			<	<	<	^	<	<
Производительность	4				<	<	^	^	<
Надежность	2					<	^	^	^
Устойчивость	1						^	^	<
Безопасность	7							<	<
Удобство использования	5								<
Проверяемость и тестируемость	1								

Рис. 14-1. Пример определения приоритетов атрибутов качества для терминала регистрации в аэропорту

В таблице вычисляются относительные оценки атрибутов. В данном примере самой важной оказывается безопасность (оценка 7), с ней тесно связаны целостность (6) и удобство использования (5). Но для успеха важны и другие факторы: плохо, когда терминал недоступен для пассажиров или когда «падает» на середине процесса регистрации, но высший приоритет не может быть у всех атрибутов качества.

Этап определения приоритетов служит двум целям. Во-первых, он помогает сосредоточиться на выявлении требований по тем атрибутам, от которых больше всего зависит успех проекта. Во-вторых, он позволяет определить, как действовать при обнаружении противоречивых требований к качеству. В примере с терминалом регистрации удалось определить потребность в достижении определенных целей в области производительности и безопасности. Эти атрибуты могут противоречить друг другу, потому что добавление дополнительных уровней безопасности может замедлить выполнение транзакций. Но так как процесс приоритизации определил, что безопасность важнее (оценка 7) производительности (оценка 4), любые противоречия нужно разрешать в пользу безопасности.

> **Внимание!** При рассмотрении атрибутов качества не пренебрегайте мнением всех заинтересованных сторон, в том числе программистов и специалистов технической поддержки. Их приоритеты качества могут сильно отличаться от приоритетов других пользователей. Приоритеты качества также могут отличаться у разных классов пользователей. При обнаружении конфликтов вскрывайте их на ранних этапах жизненного цикла разработки, чтобы разрешить их небольшими средствами.

Этап 4. Выявление конкретных ожиданий по каждому атрибуту Пользователи, как правило, высказывают свои ожидания о качестве продукта неявно, однако все же в ходе сбора информации удается выяснить основное. Хитрость заключается в том, чтобы уловить, что же стоит за их рассуждениями о том, что система должна быть простой в обращении, быстрой, надежной

или устойчивой к сбоям. Вопросы, с помощью которых выясняются невысказанные ожидания клиентов, позволяют установить и качество продукта, и критерии дизайна, что поможет разработчикам создать отличный продукт.

Большинство пользователей не знают ответов на такие вопросы, как «Каковы ваши требования к совместимости?» или «Насколько надежным должно быть программное обеспечение?». Бизнес-аналитику нужно задавать вопросы, подводящие пользователей к анализу совместимости, надежности и других атрибутов. Роксанна Миллер (Roxanne Miller, 2009) предоставила обширные списки предполагаемых вопросов, которые можно использовать при выявлении требований к качеству, кроме того в этой главе есть много примеров. При планировании встречи по выявлению требований бизнес-аналитик должен начать со списка вопросов, похожих на те, что предоставила Миллер, и отобрать только те, что важны для проекта. В качестве иллюстрации приведу несколько вопросов, которые бизнес-аналитик может задать, чтобы понять ожидания пользователя в плане производительности системы для управления размещенными инвесторами заявками на патенты:

1. Каково разумное или приемлемое время реакции на запрос стандартной заявки на патент?
2. Какое время реакции на стандартный запрос пользователи сочтут неприемлемым?
3. Сколько в среднем пользователей будут работать с системой одновременно?
4. Какое ожидается максимальное число одновременно работающих пользователей?
5. В какое время дня, недели, месяца или года нагрузка на систему выше обычной?

Отправка похожего списка вопросов участникам до встречи дает им возможность обдумать свои ответы и не отвечать «на ходу» в течение встречи. Хороший завершающий вопрос на такой встрече таков: «Есть ли что-то такое, о чем я не спросил, но что вы бы хотели обсудить?»

Возможно, стоит спросить пользователей, как они представляют себе *неприемлемые* производительность, безопасность или надежность. Это позволит определить свойства системы, которые противоречат ожиданиям пользователей о качестве, например возможность удаления файлов неполномочным пользователем (Voas, 1999). Определяя неприемлемые характеристики, попробуйте разработать тесты, чтобы реализовать эти характеристики системы на практике. Если это вам не удастся, то, вероятно, вы достигли своих целей в отношении атрибутов. Такой способ особо важен для приложений, безопасность которых критически важна: негативные изменения их надежности или безопасности аукнутся тяжелыми последствиями.

Еще одна возможная стратегия выявления требований к качеству — начать с целей по качеству, имеющихся для разрабатываемой системы у заинтересованных лиц (Alexander и Beus-Dukic, 2009). Выраженная заинтересо-

ванным лицом цель по качеству может разбиваться для выявления функциональных и нефункциональных подцелей — а, значит, требований — которые более конкретны и проще для измерения.

Этап 5. Определение качественно структурированных требований к качеству Упрощенные требования к качеству, такие как «Система должна быть дружественной по отношению к пользователю» или «Система должна быть доступна в режиме 24x7», бесполезны. Первое слишком субъективно и туманно, а второе редко бывает реалистичным или необходимым. И ни одно из них не поддается измерению. Такие требования содержат мало инструкций для разработчиков. Поэтому последний этап заключается в создании конкретных и поддающихся проверке требований на основе информации, собранной по каждому атрибуту качества. При написании требований к качеству помните мнемоническое правило SMART, то есть они должны быть конкретными (Specific), измеримыми (Measurable), достижимыми (Attainable), актуальными (Relevant) и ограниченными во времени (Time-sensitive).

Требования к качеству должны быть измеримыми, чтобы можно было добиться четкого соглашения об ожиданиях между бизнес-аналитиком, командой разработчиков и клиентами. Если это не так, мало смысла в их определении, потому что вам не удастся определить, достигли ли вы желаемой цели. Нехорошо, если тестировщику не удается протестировать требование. Следует указывать масштаб или единицы измерения для каждого атрибута и поставленной цели, а также их минимальные и максимальные значения. Система обозначений Planguage, которая описывается далее в этой главе, поможет вам в этом. Она может потребовать дополнительных обсуждений с пользователями для выявления ясных и измеримых критериев проверки требований к качеству.

Сьюзан и Джеймс Робертсон (Suzanne и James Robertson, 2013) рекомендуют включить в спецификацию каждого функционального и нефункционального требования *критерий соответствия* — «выражение требования в цифрах, показывающих стандарт, которого должен достичь продукт». Это отличный совет. Критерий соответствия описывает измеримый способ проверки правильности реализации каждого требования. Он позволяет дизайнерам выбирать решение, которое по их мнению будет соответствовать цели, а тестировщикам — оценивать результаты.

Вместо того, чтобы изобретать собственный способ документирования неизвестных требований, воспользуйтесь существующими приемами работы с требованиями. Эти приемы определяют, как писать требования того или иного типа, а также предоставляет шаблон, который можно наполнять подробной информацией, уместной в вашей ситуации. Стивен Уитхолл (Stephen Withall, 2007) предоставляет много приемов определения требований к качеству, в том числе к производительности, доступности, гибкости масштабируемости, безопасности, пользовательскому доступу и удобству установки. Приведенные советы помогут даже начинающим бизнес-аналитикам писать требования к качеству.

Определение требований по качеству

В этом разделе подробно описываются все перечисленные в таблице 14-1 атрибуты и приводятся примеры требований к качеству в различных проектах. Сьорен Лауэсен (Soren Lauesen, 2002) и Роксан Миллер (Roxanne Miller, 2009) предоставляют много дополнительных примеров корректно определенных требований к атрибутам качества. Как и с другими требованиями, рекомендуется записывать источник каждого требования к качеству и обоснования целей по качеству, если они не очевидны. Обоснование важно на случай возникновения вопросов о необходимости той или иной цели или для обоснования соответствующих затрат. Такая информация об источниках была опущена в представленных в этой главе примерах.

Внешние атрибуты качества

Внешние атрибуты качества описывают характеристики, наблюдаемые при выполнении ПО. Они сильно влияют на восприятие системы пользователями и на создающееся у пользователей мнение о ее качестве. В этой главе описаны следующие внешние атрибуты качества: доступность, удобство установки, целостность, совместимость производительность, устойчивость, защита, безопасность и удобство использования.

Доступность

Под *доступностью* (availability) понимается запланированное время доступности, в течение которого система действительно доступна для использования и полностью работоспособна. Формально доступность равна среднему времени наработки на отказ (mean time to failure, MTTF) системы, деленному на сумму среднего времени наработки на отказ и среднего времени восстановления после отказа (mean time to repair, MTTR). На доступность также влияют периоды планового технического обслуживания. Доступность тесно связана с надежностью и такой стороной возможности модификации, как легкости в эксплуатации.

Отдельные задачи более других зависят от времени, и пользователи будут страшно разочарованы (и даже разгневаны), если система не окажется доступной, когда нужно выполнить очень важную работу. Узнайте у клиентов, какой процент времени доступности им действительно необходим и есть ли периоды времени, когда доступность настоятельно необходима для бизнеса или выполнения задач, связанных с безопасностью. Для веб-сайтов, облачных приложений и приложений, с которыми работают пользователи по всему миру, требования по доступности особенно сложны и важны. Требование к доступности можно сформулировать так:

AVL-1. Система должна быть доступна как минимум на 95% по рабочим дням, с 6:00 до полуночи по местному времени и доступна как минимум на 99% по рабочим дням, с 15:00 до 18:00 по восточному поясному времени.

Как и другие приведенные здесь примеры, это требование несколько упрощено. Оно не определяет уровень производительности, который определяет, что такое *быть доступным*. Считается ли система доступной, если в режиме с плохими характеристиками в ней может работать только один пользователь сети? Скорее всего нет.

Требования к доступности иногда оговариваются в контрактной манере, как соглашение об уровне обслуживания. В случае невыполнения оговоренного уровня сервиса поставщики сервисов могут платить штраф. Такие требования должны точно определять, в чем заключается (или нет) доступность системы, и могут содержать примерно такие формулировки:

AVL-2. Плановые простои на обслуживание системы в период с 18:00 тихоокеанского времени воскресенья до 3:00 тихоокеанского времени понедельника вычитаются из вычислений для определения доступности.

Цена качества

Не указывайте 100% для таких атрибутов качества, как надежность или доступность, поскольку такие результаты недостижимы, а стремление достичь их обойдется дорого. Жизненно важные приложения, такие как системы управления воздушным движением, должны удовлетворять очень жестким — и обоснованным — требованиям. Для одной такой системы задали требование «пять девяток», то есть система должна была быть доступной 99,999% времени. То есть допустимый простой системы составлял не более 5 минут 15 секунд в год. На одно это требование пришлось четверть затрат на создание системы. Оно потребовало практически удвоить расходы на оборудование, так как требовалось резервирование, а также внести очень сложные архитектурные изменения для обеспечения горячего резерва и стратегии перехода в случае сбоя.

При выявлении требований к доступности задавайте следующие вопросы, чтобы изучить соответствующие проблемы (Miller, 2009):

- Для каких частей системы критически важно оставаться доступными?
- Каковы бизнес-последствия недоступности системы для ее пользователей?
- Если требуется регулярное обслуживание, то когда его лучше всего планировать? Как это влияет на доступность системы? Какая минимальная и максимальная продолжительность периодов обслуживания? Как во время периодов обслуживания обрабатывать попытки доступа со стороны пользователей?
- Если обслуживание должно выполняться на работающей системе, то какое влияние оно будет оказывать на доступность и как минимизировать это влияние?
- Каковы нужны уведомления пользователей о том, что система стала недоступной?

- У каких частей системы более строгие требования к доступности по сравнению с остальными частями системы?
- Какие зависимости доступности существуют между группами функциональности (например, невозможность принять платеж по кредитной карте из-за недоступности функции авторизации кредитных карт)?

Удобство установки

ПО бесполезно, пока не установлено на соответствующем устройстве или платформе. Вот несколько примеров установки ПО: загрузка приложения на смартфон или планшет, перенос ПО с компьютера на веб-сервер, обновление операционной системы, установка большой серийной системы, такой как средства планирования ресурсов предприятия, загрузка встроенного ПО в модем кабельного телевидения или установка пользовательского приложения на персональный компьютер. *Удобство установки* (installability) определяет, насколько просто правильно выполнять эти операции. Повышение удобства установки системы подразумевает сокращение времени, затрат, отвлечения пользователей, частоты возникновения ошибок, а также снижение квалификации, необходимой для выполнения установки. Удобство установки охватывает следующие операции:

- начальная установка;
- восстановление после неполной, некорректной или прерванной пользователем установки;
- переустановка той же версии;
- установка новой версии;
- возврат к предыдущей версии;
- установка дополнительных компонентов и обновлений;
- удаление ПО.

Мерой удобства установки является среднее время установки системы. Но оно зависит от множества факторов: опыта и квалификации установщика, быстродействия компьютера, на который устанавливается ПО, носителя, с которого выполняется установка (загрузка из Интернета, установка по локальной сети или с CD- или DVD-диска), ручных операций, выполняемых при установке и т. п. Группа Testing Standards Working Party предоставляет подробный список руководств и рекомендаций по требованиям к установке и установочному тестированию на веб-странице *www.testingstandards.co.uk/installability_guidelines.htm*. Далее приводятся примеры требований к установке:

INS-1. Необученный пользователь должен суметь выполнить начальную установку приложения в среднем за 10 минут.

INS-2. При установке обновленной версии приложения должна сохраниться вся настройка пользовательских параметров профиля, а также преобразована в формат данных новой версии, если это нужно.

INS-3. Программа установки должна проверять правильность загруженного установочного пакета до начала процесса установки.

INS-4. Для установки этого ПО на сервере требуются административные полномочия.

INS-3. После успешного завершения установки программа установки должна удалить все относящиеся к приложению временные, архивные, устаревшие и ненужные файлы.

А вот примеры вопросов, которые следует задавать в процессе выявления требований к установке:

- Какие установочные операции должны выполняться без нарушения пользовательского сеанса?
- Какие установочные операции потребуют перезагрузки приложения, компьютера или устройства?
- Что должно приложение выполнить после удачной — или неудачной — установки?
- Какие операции должны выполняться для подтверждения правильности установки?
- Нужна ли пользователю возможность устанавливать, удалять, переустанавливать или исправлять только избранные части приложения? Если да, то какие?
- Какие другие приложения нужно остановить перед выполнением установки?
- Какие права, разрешения и привилегии нужны установщику?
- Как должна система вести себя в случае незавершенной установки, например по причине отключения питания или прерывания пользователем?

Целостность

Целостность (integrity) подразумевает предотвращение потери информации и сохранение корректности введенной в систему информации. В требованиях к целостности нет места ошибкам: данные либо находятся в хорошей форме и защищены, либо нет. Данные должны быть защищены от таких опасностей, как случайная потеря или повреждение, вроде бы идентичные данные, которые на самом деле отличаются, физическое повреждение носителя, случайное удаление файла или перезапись данных пользователями. К рискам также относят целенаправленные атаки с прицелом на умышленное повреждение или кражу данных. Безопасность иногда считается подмножеством целостности, потому что некоторые требования к безопасности призваны предотвратить неправомочный доступ неавторизированных пользователей. Требования к целостности должны гарантировать, что данные, полученные из других систем, совпадают с тем, что было отправлено, и наоборот. Иногда целью атаки становятся исполняемые файлы ПО, поэтому нужно также защитить их целостность.

Целостность данных также подразумевает корректность и правильное форматирование данных (Miller, 2009). Сюда относятся такие вопросы, как форматирование полей дат, разрешение ввода в поля только правильного типа и длины данных, гарантирование, что элементы данных содержат корректные значения, проверка правильности ввода в одно поле, когда в другом поле введено определенное значение и т. п. Далее приводятся примеры требований к целостности:

INT-1. После резервного копирования файла система должна сверить архив с оригиналом и в случае несовпадения сообщить об этом.

INT-2. Система должна защищать от неправомерного добавления, удаления или изменения данных.

INT-3. Система Chemical Tracking System должна проверять корректность закодированных химических структур, импортированных из сторонних средств моделирования таких структур.

INT-4. Система должна ежедневно проверять, что исполняемые модули приложения не подверглись изменению путем добавления неполномочного кода.

Вот некоторые факторы, которые надо учесть при обсуждении требований к целостности (Withall, 2007):

- Обеспечение того, чтобы изменения в данных выполнялись целиком или вообще не выполнялись. Это может означать отмену изменений данных, если в середине процесса возникнут неполадки.
- Обеспечение постоянства изменений внесенных в данные.
- Координация изменений, вносимых во многие хранилища данных, особенно если изменения должны выполняться одновременно (скажем, на многих серверах) и в определенное время (скажем в 12:00 Гринвичского времени 1 января в нескольких местах).
- Обеспечение физической безопасности компьютеров и внешних устройств хранения.
- Выполнение архивирования данных. (С какой частотой? Автоматически и/или по требованию? В каких файлах или базах данных? На какие носители? Со сжатием и проверкой или без?)
- Восстановление данных из архива.
- Архивирование данных: каких данных, когда архивировать, насколько долго и с какими требованиями по удалению.
- Защита данных, хранимых или заархивированных в облаке, от людей, у которых не должно быть доступа к ним.

Совместимость

Совместимость (interoperability) определяет, насколько система готова обмену данными с другими программными системами и к интеграции с внешними аппаратными устройствами. Чтобы оценить совместимость, необходимо знать, какие другие приложения пользователи будут применять совмест-

но с вашим продуктом и обмен каких данных предполагается. Пользователи Chemical Tracking System привыкли рисовать химические структуры, с помощью нескольких программ, поэтому они выдвинули следующее требование к совместимости:

IOP-1. Chemical Tracking System должна иметь возможность импортировать любые допустимые химические структуры из программ ChemiDraw (версии 13.0 или более ранней) и MarvinSketch (версии 5.0 или более ранней).

Можно было бы указать данное требование как требование к внешнему интерфейсу и определить стандартные форматы информации, которые может импортировать Chemical Tracking System. Можно также было определить несколько функциональных требований, относящихся к операции импорта. Выявление и документирование таких требований важнее, чем их классификация.

> **Внимание!** Не храните одно требование в нескольких местах, даже если оно логически там уместно. Это лишняя причина рассогласования, если вы измените, к примеру, требование к совместимости, но забудете изменить ту же информацию, которую вы также зафиксировали как функциональные требования или требования к внешнему интерфейсу.

Требования к совместимости могут диктовать применение стандартных форматов обмена данными для обеспечения взаимодействия с другими программными системами. Таким требованием к Chemical Tracking System было следующее:

IOP-2. Chemical Tracking System должна иметь возможность импортировать любые химические структуры в кодировке SMILES (Simplified Molecular Input Line Entry Specification, то «есть спецификация упрощенного представления молекул в строке ввода»).

Рассматривая систему с точки зрения атрибутов качества, можно обнаружить некоторые ранее не выявленные требования. Пользователи не указали данную потребность при обсуждении внешнего интерфейса или функциональности системы. Как только бизнес-аналитик задал вопрос о других системах, с которыми придется взаимодействовать Chemical Tracking System, сторонник продукта немедленно упомянул две программы для рисования химических структур.

Вот примеры вопросов, которые можно использовать при изучении требований к совместимости:

- С какими другими системами должна взаимодействовать эта система? Какими сервисами или данными они должны обмениваться?
- Какие стандартные форматы данных нужны для обмена с другими системами?
- Какие особые аппаратные компоненты должны взаимодействовать с системой?
- Какие сообщения или коды система должна получать и обрабатывать от других систем и устройств?

- Какие стандартные протоколы связи необходимы для обеспечения совместимости?
- Как внешние обязательные требования к совместимости должна удовлетворять система?

Производительность

Производительность (performance) один из атрибутов качества, о которых пользователи часто упоминают по делу и без дела? Производительность характеризует скорость реакции системы на пользовательские запросы и действия, но, как видно из табл. 14-2 она охватывает намного больше. Уитхолл (Withall, 2007) предлагает приемы для определения некоторых из этих классов требований к производительности.

Низкая производительность раздражает пользователей, которые ожидают вывода на экран результата запроса базы данных. Но проблемы производительности также ставят под удар безопасность, например из-за необходимости перегрузки системы управления процессами реального времени. Жесткие требования к производительности сильно влияют на стратегию разработки ПО и выбор оборудования, подходящего для операционной среды. Все пользователи хотят, чтобы их приложение запускалось моментально, но реальные требования к производительности различаются для функции проверки орфографии в программе подготовки текстов и для радиолокационной системы наведения ракеты. Удовлетворение требований к производительности может быть сложным, потому что оно зависит от слишком большого числа внешних факторов, таких как быстродействие используемого компьютера, сетевых подключений и других аппаратных компонентов.

Табл. 14-2. Некоторые особенности производительности

Измерение производительности	Пример
Время реакции	Время до отображения веб-страницы в секундах
Пропускная способность	Число обрабатываемых в секунду транзакций с кредитными картами
Емкость данных	Максимальное число записей, хранимых в базе данных
Динамическая емкость	Максимальное число одновременно обслуживаемых пользователей на веб-сайте социальной сети
Предсказуемость систем реального времени	Жесткие требования к временны*м* характеристикам системы управления полетом на самолете
Время доступа	Задержки времени в ПО записи музыки
Поведение в режиме ограниченной функциональности или в условиях перегрузки	Стихийное бедствие приводит к массовому увеличению звонков в службу спасения

В процессе документирования требований к производительности обоснуйте их необходимость, чтобы помочь разработчикам принять правильные решения, касающиеся дизайна. Например, из-за жестких требований к времени отклика базы данных разработчики могут обеспечить зеркальное отображение базы данных в нескольких географических местах. Укажите количество транзакций в секунду, которое будет поддерживаться, время отклика и временные зависимости для систем реального времени. Вы также можете указать требования к памяти и свободному пространству на диске, к количеству одновременно работающих пользователей или к максимальному количеству строк в таблицах баз данных. Пользователи и бизнес-аналитики могут не обладать всей информацией, поэтому нужно сотрудничать с различными заинтересованными лицами над выявлением как можно большего числа аспектов требований к производительности. Далее приводятся примеры требований к производительности:

PER-1. Авторизация при запросе на получение денег в банкомате должна занимать не более 2,0 секунд.

PER-2. Антиблокировочная система тормозов должна информировать о скорости колеса каждые 2 миллисекунды с разбросом не более 0,1 миллисекунд.

PER-3. Полная загрузка веб-страницы по подключению с быстродействием 30 Мбит/с должна занимать в среднем 3 секунды.

PER-4. После заключения сделки как минимум 98% времени торговая система должна обновлять состояние транзакции за одну секунду.

Производительность — внешний атрибут качества, потому что его можно наблюдать только во время работы программы. Она тесно связана с таким внутренним атрибутом качества, как эффективность, которая сильно влияет на наблюдаемую пользователями производительность.

Надежность

Вероятность работы ПО без сбоев в течение определенного периода времени называется *надежностью* (reliability) (Musa, 1999). Проблемы с надежностью иногда происходят из-за ввода неправильной информации, ошибок в самом коде, компонентов, недоступных, когда они нужны, а также из-за отказов оборудования. Устойчивость и доступность тесно связаны с надежностью. Для измерения надежности ПО используют такие показатели, как процент успешно завершенных операций и среднее время наработки на отказ (Mean Time Between Failures, или MTBF) и максимальная приемлемая вероятность отказа за определенный период времени. Определите количественные требования к надежности, основываясь на том, насколько серьезными окажутся последствия сбоя и оправданы ли затраты на повышение надежности. Системы, для которых требуется высокая надежность, следует также проектировать с высокой степенью проверяемости, чтобы облегчить выявление недостатков, отрицательно влияющих на надежность.

Однажды моя команда разрабатывала ПО для управления лабораторным оборудованием, предназначенным для опытов с редкими дорогостоящими химикатами, длящихся целый день. Пользователям требовался программный компонент, который обеспечил бы надежность проведения экспериментов. Другие системные функции, такие, как периодическая запись в журнал данных о температуре, были не столь важными. Требования к надежности для данной системы звучали так:

REL-1. Не более 5 из 1000 начатых экспериментов могут быть потеряны из-за сбоев ПО.

Одни отказы системы приводят к серьезным последствиям, другие не оказывают такого разрушающего воздействия. Отказ может заставить пользователя перезапустить приложение и восстановить данные, которые он сохранял. Это неприятно, но не катастрофично. Последствия сбоев, приводящих к потере или повреждению данных, например сбой фиксации транзакции в базе данных, намного серьезнее. Лучше предотвращать ошибки, чем искать их и пытаться устранить.

Как и многие другие атрибуты качества, надежность — запаздывающий индикатор, то есть нельзя узнать, достигли ли заданного качества, пока система не поработает какое-то время. Посмотрите на следующий пример:

REL-2. Среднее время между отказами устройства чтения карт должно не превышать 90 дней.

Нет способа узнать, выполняется ли это требование в системе, пока не пройдет 90 дней. Но можно точно узнать, что система *не соответствует* требованию, если за 90 дней устройство откажет более двух раз.

Вот примеры вопросов, которые надо задавать представителям пользователей во время выявления требований к надежности:

- Как вы будете определять, достаточно ли надежна система?
- Каковы должны быть последствия сбоя при выполнении определенных операций в системе?
- Что вы будете считать критическим отказом, а что — всего лишь досадной неприятностью?
- При каких условиях сбой может иметь серьезные последствия для ваших бизнес-операций?
- Никому не нравится сталкиваться со сбоем системы, но есть ли какие-то части системы, которые совершенно точно должны быть чрезвычайно надежными?
- Если система станет неработоспособной, сколько времени вы сможете обойтись без нее, пока это заметно не скажется на ваших бизнес-операциях?

Понимание требований к надежности позволяет архитекторами, дизайнерам и разработчикам предпринимать меры, которые они посчитают уместными для достижения нужной надежности. С точки зрения требований один из способов сделать систему одновременно надежной и устойчивой — указать исключительные условия и что нужно делать при их наступлении. Плохая

обработка исключений может создать у пользователей ощущение низкой надежности и удобства использования. Пользователей очень раздражают вебсайты, на которых очищается вся введенная пользователем информация при первой же ошибке ввода значения в очередном поле. Никакой пользователь не назовет это поведение приемлемым. Разработчики могут сделать системы надежнее, применяя приемы «безопасного программирования», например проверку вводимых значений на корректность и подтверждение успешности завершения операций записи на диск.

Устойчивость

Однажды клиент компании, разрабатывающей устройства для измерений, пожелал, чтобы их следующий продукт «был, как танк». Сотрудникам компании так понравилось сравнение, что они ввели в обиход новый, слегка ироничный атрибут качества — «как танк». Его стали применять, когда речь шла об устойчивости. Под устойчивостью (robustness) понимают уровень, до которого система продолжает корректно выполнять свои функции, несмотря на неверный ввод данных, недостатки подключенных программных или аппаратных компонентов или неожиданные условия работы. Устойчивое ПО легко восстанавливается после различных проблем и «не замечает» ошибок пользователей. Оно восстанавливается после внутренних сбоев, не оказывая отрицательного влияния на работу конечных пользователей. Программные ошибки обрабатываются так, что пользователь воспринимает это как разумное, а не раздражающее поведение. Устойчивость также называют *устойчивостью к сбоям* (fault tolerance) (перехватываются и корректируются ли ошибки ввода пользователем?), *выживаемостью* (survivability) (перенесет ли без последствий камера падение с определенной высоты?) и *восстанавливаемостью* (recoverability) (сможет ли компьютер восстановить нормальную работу после отключения питания во время операции по обновлению операционной системы?).

Выясняя требования к устойчивости работы ПО, спросите пользователей, какие ошибочные ситуации возможны при работе с системой и как система должна на них реагировать. Подумайте о возможных способах обнаружения возможных ошибок, которые могут вести к отказу системы, уведомления о них пользователя и восстановления после отказа. Убедитесь, что вы точно понимаете, когда должна корректно завершиться одна операция (например, подготовка данных для передачи), чтобы могла начаться следующая (например, передача данных в другую систему). Вот один из примеров требования к устойчивости:

ROB-1. Если при работе с редактором произошел сбой и пользователь не успел сохранить файл, то редактор должен восстановить все изменения, внесенные раньше, чем за минуту до сбоя, при следующем запуске программы данным пользователем.

Такое требование может заставить разработчика реализовать механизм создания контрольных точек или автосохранения для минимизации поте-

ри данных вместе с функциональностью проверки при запуске наличия сохраненных данных и восстановления содержимого файла. Но в требование к устойчивости не нужно точно описывать точный механизм. Оставьте эти детали разработчикам.

Сам виноват

Во время работы над этой главой я получил возможность получить некоторый опыт в области устойчивости. Я запустил печать черновой версии главы и отправил компьютер в спящий режим до завершения печати, надеясь, что все данные уже переправлены на принтер. Но это было не так. Как должен диспетчер печати обработать ошибку, когда я верну компьютер из спящего режима? Должен ли он завершить работу и печатать оставшуюся часть файла, перепечатать все с начала или что? В итоге он перепечатал все задание, хотя я бы предпочел, чтобы он продолжил печать с того места, где остановился. Получился небольшой перерасход бумаги, но по крайней мере диспетчер восстановился после ошибки пользователя и продолжил работу.

Несколько лет назад я занимался разработкой повторно используемого программного компонента Graphics Engine, который преобразовывал файлы с графическими схемами и выводил изображение на назначенное устройство вывода. Несколько приложений, которым было необходимо создавать графики, обращались к Graphics Engine. Поскольку разработчики не могли контролировать, какие именно данные приложения передают в Graphics Engine, важным атрибутом качества была названа устойчивость. Одно из наших требований звучало так:

ROB-2. Для всех параметров, описывающих графики, должны быть указаны значения по умолчанию, которые Graphics Engine будет использовать в случае, если данные входного параметра указаны неверно или отсутствуют.

Выполнение этого требования позволит избежать сбоя программы, если, например, приложение запрашивает стиль линии, который не поддерживается. Graphics Engine задействует значение по умолчанию — сплошная линия — и продолжит работу. Тем не менее это можно рассматривать как сбой ПО, поскольку конечный пользователь не получил желаемый результат. Однако такой подход снизил серьезность последствий сбоя — вместо краха программы получен неправильный цвет, что является примером отказоустойчивости.

Защита

Требования к *защите* (safety) подразумевают, что система должна стараться не нанести ранение или другой вред людям и имуществу (Leveson, 1995; Hardy, 2011). Требования к защите могут регулироваться требованиями регулирующих органов и другими бизнес-правилами, юридические вопросы и вопросы сертификации могут быть связаны с выполнением таких требований.

Требования к защите часто формулируются в форме условий или действий, которые в системе не должны происходить.

Люди редко получают ранения от взрывающихся электронных таблиц, однако управляемые программами устройства могут нанести ощутимый вред человеку. Даже в некоторых исключительно программных приложениях есть очевидные требования к защите. В приложении, позволяющем пользователям заказывать еду в кафе, может быть предусмотрено следующее требование к защите:

SAF-1. У пользователя должна быть возможность увидеть список всех ингредиентов каждого блюда, причем ингредиенты, известные тем, что могут вызывать аллергическую реакцию у 0,5% населения Северной Америки, должны быть выделены особо.

Такие возможности веб-браузера, как родительский контроль, который закрывает доступ к определенным функциям и URL-адресам, может рассматриваться как удовлетворением требований к защите или безопасности. Но чаще приходится видеть требования к защите аппаратных систем, например:

SAF-2. Если температура в корпусе реактора повышается быстрее, чем 5°C в минуту, система управления химическим реактором данных должна выключить подогрев и дать предупредительный сигнал оператору.

SAF-3. Устройство терапевтического облучения должна выполнять облучение, только при условии наличия соответствующего фильтра.

SAF-4. Система должна в течение секунды прекратить все операции, если измеренное в резервуаре давление превышает 90% заданного максимального давления.

В процессе выявления требований по защите может потребоваться проинтервьюировать специалистов предметной области, которые хорошо знакомые с рабочей средой, или с людьми, которые много думали над рисками проекта. Вот примеры вопросов, которые можно задать:

- При каких условиях использование этого продукта может нанести вред человеку? Как система может обнаружить эти условия? Как нужно реагировать на них?
- Какая максимально разрешенная частота отказов, способная вызывать повреждения?
- Какие виды отказов могут приносить вред людям или имуществу?
- Какие действия оператора могут непреднамеренно нанести вред людям или имуществу?
- Есть ли какие-то режимы работы, которые могут представлять риск для людей или имущества?

Безопасность

Безопасность (security) связана с блокировкой неавторизированного доступа к системным функциям или данным, предотвращением потерь от атак вредоносного кода и т. п. Целостность очень важна для интернет-приложений.

Пользователи систем электронной коммерции хотят обезопасить данные своих кредитных карточек. Посетители веб-сайтов не желают, чтобы личная информация о них или список посещаемых ими сайтов использовались не по назначению, а поставщики услуг доступа к Интернету хотят защититься от атак типа «отказ в обслуживании» и прочих хакерских атак. Как и в требованиях к целостности, в требованиях к безопасности нет места ошибкам. Вот ряд факторов, которые надо проанализировать при выявлении требований к безопасности:

- уровни авторизации и привилегий пользователей (рядовой пользователь, гость, администратор) и управление доступом пользователей (матрица ролей и разрешений, пример которой показан на рис. 9-2, может оказаться очень кстати);
- идентификация и проверка подлинности пользователей (правила написания паролей, частота изменений пароля, вопросы безопасности, процедуры восстановления забытого имени пользователя или пароля, биометрическая идентификация, блокировка учетных записей после неудачных попыток доступа, неопознанный компьютер);
- конфиденциальность данных (кто может создавать, просматривать, изменять, копировать, распечатывать и удалять какую информацию);
- умышленное уничтожение, повреждение или кража данных;
- защита от вирусов и червей, троянцев, шпионских программ, руткитов и другого вредоносного ПО;
- брандмауэр и другие вопросы сетевой безопасности;
- шифрование безопасных данных;
- аудит выполняемых операций и попыток доступа.

Далее приводится несколько примеров требований к безопасности. Легко увидеть, как можно построить тесты для проверки правильности реализации этих требований.

SEC-1. Система должна заблокировать учетную запись пользователя после четырех неудачных попыток входа в систему за пять минут.

SEC-2. Система должна регистрировать в журнале все попытки доступа к защищенным данным пользователей, не обладающих достаточным уровнем полномочий.

SEC-3. Пользователь должен изменить временный пароль, назначенный специалистом по безопасности, на новый пароль сразу же после первого успешного входа в систему с использованием временного пароля.

SEC-4. Дверь, разблокированная в результате успешного считывания бейджа безопасности, должна оставаться разблокированной на протяжении 8,0 с разбросом в полсекунды.

SEC-5. Резидентное противовирусное ПО должно размещать в карантине весь интернет-трафик, в котором содержатся известные или подозрительные сигнатуры вирусов.

SEC-6. Магнитометр должен обнаруживать не менее 99,9% запрещенных объектов с долей ложных срабатываний, не превышающей 1%.

Требования к безопасности часто основываются на бизнес-правила, таких как корпоративные политики безопасности, как показано в этом примере:

SEC-7. Только пользователи, обладающие привилегиями уровня Аудитор, должны иметь возможность просматривать транзакции клиентов.

Постарайтесь не формулировать требования к безопасности, включая в них вложенные ограничения дизайна. Пример — определение паролей в качестве механизма управления доступом. Реальное требование должно предусматривать ограничение доступа к системе неполномочных пользователей; пароли — всего лишь один из способов (хотя и самый распространенный) выполнения этой задачи. Основанное на выбранном подходе к идентификации пользователей, это базовое требование к целостности повлияет на конкретные функциональные требования, которые реализуют метод проверки подлинности.

Вот ряд вопросов, которые можно задать при выявлении требований к безопасности:

- Какие конфиденциальные данные нужно защитить от неправомочного доступа?
- Кто уполномочен просматривать конфиденциальные данные? И кто конкретно не уполномочен?
- При каких бизнес-условиях или в каких временны*х* рамках полномочным пользователям предоставляется доступ к функциональности?
- Какие проверки должны выполняться для проверки того, что пользователь выполняет приложение в безопасной среде?
- Как часто должна антивирусная программа выполнять сканирование на предмет обнаружения вредоносного ПО?
- Должен ли применяться какой-то особый метод проверки подлинности пользователей?

Удобство использования

Удобство использования (usability) подразумевает мириады факторов, которые люди в разговорной речи называют *дружественностью к пользователю, простоте использования* и *эргономической проработкой*. Но в лексиконе аналитиков и разработчиков нет термина «дружественное ПО», они говорят о ПО, которое спроектировано для эффективного и необременительного использования. Удобство и простота использования измеряется усилиями, требуемыми для подготовки ввода данных, эксплуатации и вывода конечной информации.

В последнее время вышло несколько книг, посвященных разработке удобных в использовании программных систем (например, Constantine и Lockwood, 1999; Nielsen, 2000; Lazar, 2001; Krug, 2006; Johnson, 2010). Удобство использования охватывает несколько областей помимо очевидной простоты

использования — легкость изучения, запоминаемость, предотвращение ошибок, поведение и восстановление, эффективность взаимодействия, специальные возможности и эргономика. Между этими категориями могут возникать противоречия. Например, легкость изучения может конфликтовать с простотой использования. Возможности, которые дизайнер предусмотрел для удобства новых или редко работающих с продуктом пользователей, могут раздражать опытного пользователя, который точно знает, что хочет, и нуждается в максимально эффективном выполнении задачи. У разных функций приложения также могут быть разные цели с точки зрения удобства использования. Может быть важным очень эффективный ввод данных, а также возможность быстро понять, как создать нестандартный отчет. Табл. 14-3 иллюстрирует некоторые приемы реализации удобства использования. Как видите, возможны конфликты в рамках определенных классов пользователей, если оптимизация одного аспекта удобства выполняется в ущерб другому.

> **Внимание!** Главной задачей удобства использования — впрочем, как и других атрибутов качества — является баланс удобства для всего спектра пользователей, а не какой-то одной их части. Это означает, что некоторые пользователи будут не слишком довольны результатом. Возможности пользовательской настройки могут повысить привлекательность приложения.

Табл. 14-3. Возможные приемы дизайна легкости изучения и использования

Легкость изучения	Простота использования
Подробные подсказки	Быстрые клавиши
Мастера	Развитые настраиваемые меню и панели инструментов
Видимые команды меню	Много вариантов доступа к одной и той же функции
Содержательные сообщения на понятном для пользователя языке	Автозавершение ввода
Справочная система и всплывающие подсказки	Автоисправление ошибок
Похожесть на другие аналогичные системы	Запись макросов и возможности создания сценариев (скриптов)
Отображение ограниченного числа команд и виджетов	Возможность переносить информацию из предыдущей транзакции
	Автоматическое заполнение полей форм
	Интерфейс командной строки

Как и с другими атрибутами качества, можно измерять разные аспекты «дружественности по отношению к пользователю». Вот некоторые признаки удобства использования:

- среднее время, необходимое конкретному типу пользователей, для правильного выполнения определенной задачи;
- сколько транзакций может правильно выполнить пользователь за заданный период времени;
- какую долю набора задач пользователь может правильно выполнить без посторонней помощи;
- сколько ошибок делает пользователь при выполнении задачи;
- сколько попыток требуется пользователю для выполнения определенной задачи, например поиска определенной функции, спрятанной в глубинах меню;
- задержка или время ожидания при выполнении задачи;
- число операций (щелчков мыши, нажатий клавиш, жестов на сенсорном экране), необходимых для получения нужной информации или выполнения задачи.

Просто скажи мне, что не так

Просчеты в области удобства использования могут очень раздражать. Недавно я пытался сообщить об ошибке, используя форму обратной связи на одном веб-сайте. Я получил сообщение об ошибке с формулировкой «специальные символы запрещены», но ничего не говорилось, какие конкретно символы в моем тексте не устраивают веб-сайт. Ясно, что программа знала, какие символы «плохие», потому что сумела их найти. Сообщение об ошибке общего характера вместо конкретного совета не позволило разрешить проблему. В конце концов я догадался, что программу не устраивают кавычки в моем сообщении. Я никогда не думал, что кавычки можно считать специальными символами, и вообще термин «специальные символы» неясный и неоднозначный. Чтобы помочь разработчикам определить, как лучше всего удовлетворить ожидания пользователей в области удобства использования, бизнес-аналитик должен записать конкретные требования к удобству использования, а разработчики должны обеспечить отображение точных и внятных сообщений об ошибках в нужных местах.

Чтобы оценить ожидания пользователей в области удобства программы, аналитики требований для Chemical Tracking System задавали представителям пользователей такие вопросы, как «Сколько шагов вы готовы выполнить в процессе заказа химикатов?» и «Сколько времени вы готовы отвести на запрос?». Все это — несложные исходные точки для определения многих характеристик, которые могут сделать ПО удобным в работе. При обсуждении этого атрибута удается выявить параметры, поддающиеся измерению, например:

USE-1. Пользователь, прошедший соответствующую подготовку, должен в 95% времени иметь возможность выбрать требуемый химикат из каталога поставщика в среднем за три и максимум за пять минут.

Узнайте, должна ли новая система соответствовать каким-либо стандартам или соглашениям, касающимся пользовательского интерфейса, и должен ли последний быть совместим с другими часто используемыми системами. Вы можете сформулировать такое требование следующим образом:

USE-2. Все функции меню File должны иметь определенные быстрые клавиши, нажимаемые одновременно с Ctrl. Командам меню, которые также присутствуют в Microsoft Word, должны соответствовать те же быстрые клавиши, что и в Word.

Такое единообразие в работе позволяет избежать неприятных ошибок, когда пальцы по привычке нажимают клавиши, которые имеют другое значение в приложении, которые вы используете нечасто. Простота обучения также поддается исчислению и измерению:

USE-3. 95% химиков, которые прежде никогда не использовали Chemical Tracking System, должны не более чем 15 минут разобраться, как правильно заказать химикат.

Тщательное определение требований ко всем сторонам удобства использования поможет дизайнерам сделать выбор, который отделяет пользователей, с удовольствием работающих с продуктом, от тех, что используют продукт с неодобрительным выражением на лице или, хуже того, отказываются использовать продукт.

Внутренние атрибуты качества

Внутренние атрибуты качества нельзя наблюдать напрямую во время выполнения ПО. Это свойства, наблюдаемые разработчиками или специалистами по поддержке при изучении дизайна или кода при их модификации, повторном использовании или переносе на другую платформу. Внутренние атрибуты могут косвенно влиять на восприятие клиентом качества продукта, если в последующем оказывается сложным добавлять новую функциональность или из-за неэффективности внутренних механизмов, которые приводят к падению производительности. В этом разделе описываются атрибуты качества, которые в первую очередь важны для архитекторов, разработчиков, сотрудников отдела обслуживания и других технических специалистов.

Эффективность

Эффективность тесно связана с внешними атрибутами качества производительности. *Эффективность* (efficiency) — показатель того, насколько хорошо система использует производительность процессора, место на диске, память или полосу пропускания соединения. Если система тратит слишком много доступных ресурсов, пользователи заметят снижение производительности.

Эффективность — а значит, и производительность — основная движущая сила архитектуры, определяющая выбор дизайнера при распределении функций среди компонентов системы. Проблемы производительности ставят под удар достижения в области других атрибутов качества. Определите

минимальную конфигурацию оборудования, при которой удается достичь заданных эффективности, пропускной способности и производительности. Чтобы обеспечить запас на случай непредвиденных условий и последующий рост (это влияет на масштабируемость), можно воспользоваться такой формулировкой:

EFF-1. Как минимум 30% ресурсов процессора и оперативной памяти, доступной приложению, не должно использоваться в условиях запланированной пиковой нагрузки.

EFF-2. Система должна отображать предупреждение оператору, когда нагрузка достигает 80% максимальной плановой мощности.

Типичные пользователи не формулируют требования к эффективности в таких технических терминах. Максимум, на что они способны, так это упомянуть время отклика или заполнение пространства на диске. Дело бизнес-аналитика — задать вопросы, которые выявят ожидания пользователей о приемлемом снижении производительности, возможных пиках нагрузки и ожидаемом росте. Вот примеры таких вопросов:

- Какое число одновременно работающих пользователей ожидается сейчас и в будущем?
- Насколько могут ухудшиться время отклика и другие показатели производительности, прежде чем это начнет серьезное отрицательно сказываться на работе пользователей и бизнес-операциях?
- Сколько операций система должна выполнять одновременно в нормальных и экстремальных условиях работы?

Возможность модификации

Возможность модификации (modifiability) показывает, насколько просто разобраться в дизайне и коде ПО, изменять его и расширять. Она охватывает несколько других атрибутов качества, которые связаны с разными формами поддержки ПО, как показано в табл. 14-4. Она также тесно связана с проверяемостью. Если разработчики ожидают, что в продукт будет много вноситься изменений, они могут выбрать дизайн, обеспечивающий максимальную возможность модификации. Широкие возможности модификации крайне важны для продуктов, подвергающимся частым изменениям, и тех, что создаются поэтапно или итеративно.

Табл. 14-4. Некоторые особенности возможности модификации

Тип поддержки	Аспекты возможности модификации	Описание
Корректировка	Удобство эксплуатации, понятность	Исправление дефектов
Улучшение	Гибкость, расширяемость и дополняемость	Улучшение и модификация функциональности для удовлетворения новых бизнес-потребностей и требований

Табл. 14-4. (*окончание*)

Тип поддержки	Аспекты возможности модификации	Описание
Адаптация	Удобство эксплуатации	Модификация системы для работы в другой рабочей среде без добавления новых возможностей
Текущее обслуживание	Пригодность к обслуживанию	Исправление дефектов, обслуживание или ремонт устройств в их рабочей среде

Возможность модификации измеряют, используя такие термины, как среднее время, требуемое для добавления новой функции или разрешения проблемы, и процент корректных исправлений. Для Chemical Tracking System одно из требований к возможности модификации сформулировано таким образом:

MOD-1. Опытный программист, занимающийся техническим обслуживанием ПО, должен уметь обновлять существующие отчеты в соответствии с изменениями в положениях федерального правительства в области химии, затратив на разработку не более 10 рабочих часов.

При работе над проектом Graphics Engine мы понимали, что нам придется часто вносить исправления в ПО, чтобы оно соответствовало потребностям пользователей. Будучи опытными разработчиками мы приняли правила дизайна, которые требовали от разработчиков писать код так, чтобы программу было проще понимать, а, значит, и поддерживать. Вот одно из этих правил:

MOD-2. Вложенность вызываемых функций не должна превышать два уровня.

Такие правила нужно формулировать осторожно, чтобы не провоцировать разработчиков и не заставлять их действовать формально, выполняя букву, но не суть задачи. Бизнес-аналитик должен работать с программистами, занимающимися техническим обслуживанием, чтобы понять, какие качества исходного кода облегчат им внесение изменений или исправление недостатков.

Для аппаратных устройств со встроенным ПО часто имеются требования к возможности модификации. Одни из них относятся к выбору дизайна ПО, в то время как другие влияют на дизайн оборудования. Например, последнее можно сформулировать так:

SUP-1. Конструкция принтера должна позволять сертифицированному ремонтнику заменить модуль сканнера не более чем за 10 минут.

Требования к обслуживанию могут помогать сделать жизнь пользователя легче, как в следующем примере:

SUP-2. Принтер должен отображать сообщение об ошибке, если новые картриджи были вставлены не в те гнезда.

Переносимость

Мерой *переносимости* (portability) можно считать усилия, необходимые для перемещения ПО из одной операционной среды в другую. Некоторые практики считают интернационализацию и возможность локализации продукта частью его переносимости. Приемы разработки ПО, которые делают легким его перенос, очень схожи с теми, что применяют, чтобы сделать ПО многократного используемым. Переносимости становится все важнее ввиду того, что приложения должны работать во многих средах, таких как Windows, Mac и Linux, а также в iOS и Android и на разных компьютерах, планшетах и смартфонах. Требования к переносимости данных также важны.

Цели переносимости должны определить те части продукта, которые необходимо легко перемещать в другие среды, и описать эти целевые среды. Один продукт для анализа химикатов работал в двух очень разных средах. Одна версия работала в лаборатории, где химик со званием кандидата химических наук использовал ее для управления несколькими аналитическими инструментами. Второй экземпляр работал на мобильном устройстве, используемом в поле, в частности на нефтепроводе, человеком, с более низкой квалификацией. Базовые возможности экземпляров в целом одинаковые. Такой продукт должен изначально проектироваться с расчетом на работу в обеих средах с минимальными доработками со стороны разработчиков. Зная ожидания клиентов по переносимости, разработчики смогут выбрать способы разработки, которые позволят повысить это качество продукта. Далее приводятся примеры требований к переносимости:

POR-1. Модификация версии приложения для iOS с тем, чтобы оно могло работать на устройствах под управлением Android, должно требовать изменения не более 10% исходного кода.

POR-2. У пользователя должна быть возможность переносить закладки браузера между Firefox, Internet Explorer, Opera, Chrome и Safari.

POR-3. Средство миграции платформы должно переносить индивидуализированные профили пользователей без каких-либо усилий со стороны пользователей.

При изучении переносимости могут оказаться кстати следующие вопросы:

- На каких других платформах должно работать ПО сейчас и в будущем?
- Какие части продукта должны разрабатываться с расчетом на более высокую переносимость по сравнению с другими его частями?
- Какие файлы данных, программные компоненты и другие элементы системы должны быть переносимыми?
- Какие другие атрибуты качества могут пострадать при реализации переносимости в системе?

Возможность повторного использования

Возможность повторного использования (reusability) показывает относительные усилия, необходимые для преобразования программных компонентов

с целью их дальнейшего применения в других приложениях. Повторно используемое ПО должно быть модульным, качественно задокументированным, не зависеть от конкретных приложений и операционной среды, а также обладать некоторыми универсальными возможностями. Многие артефакты проекта могут использоваться повторно, в том числе требования, архитектуры, дизайн, код, тесты, бизнес-правила, модели данных, описания классов пользователей, профили заинтересованных лиц и термины словаря терминов (см. главу 18). Возможность повторного использования ПО обеспечивается тщательным описанием требований и дизайна, жестким соблюдением стандартов программирования, наличием набора регрессионных тестов и применением стандартной библиотеки повторно используемых компонентов.

Цели многократного использования сложно количественно измерить. Укажите, какие элементы новой системы необходимо спроектировать таким образом, чтобы упростить их повторное применение, или укажите библиотеки компонентов многократного использования, которые необходимо создать дополнительно к проекту. Например:

REU-1. Функции ввода химических структур должны быть спроектированы так, чтобы их удавалось повторно использовать на уровне объектного кода в других приложениях.

REU-2. Как минимум 30% архитектуры приложения должно состоять из одобренных архитектур повторного использования.

REU-3. Алгоритмы определения цены должны повторно использоваться будущими приложениями для управления складом.

В процессе изучения требований к повторному использованию можете задать следующие вопросы:

- Какие существующие требования, модели, компоненты дизайна, данные или тесты можно повторно задействовать в данном приложении?
- Какая функциональность, имеющаяся в связанных приложениях, удовлетворяет некоторым требованиям к данному приложению?
- Какие части этого приложения хорошо подходят для повторного использования в других местах?
- Какие особые действия нужно предпринять, чтобы части данного приложения можно было бы использовать повторно?

Масштабируемость

Масштабируемость (scalability) определяет способность приложения расти и обслуживать больше пользователей, данных, серверов, географических точек, транзакций, сетевого трафика, операций поиска и других сервисов без снижения производительности и корректности. Масштабируемость предъявляет требования к оборудованию и ПО. Масштабирование системы вверх подразумевает приобретение более быстрых компьютеров, увеличение памяти или дискового пространства, добавление серверов, зеркальное отображение баз данных или увеличение пропускной способности сети. К программным

способам относятся распределение вычислений на несколько процессоров, сжатие данных, оптимизация алгоритмов и другие приемы оптимизации производительности. Масштабируемость связана с возможностью модификации и устойчивостью, потому что один из аспектов надежности — поведение системы при приближении или превышении границы возможностей системы. Далее приводится несколько примеров требований к масштабируемости:

SCA-1. Надо обеспечить возможность увеличения пропускной способности аварийной телефонной системы с 500 до 2500 звонков в день в течение 12 часов.

SCA-2. Веб-сайт должен уметь справляться с 30-процентным ростом частоты запросов страниц в квартал на протяжении как минимум двух лет без ощутимого пользователями снижения производительности.

SCA-3. Система распространения должна поддерживать до 20 новых складских комплексов.

Бизнес-аналитик может не совсем понимать планы будущего роста конкретного приложения. В этом случае он должен поработать с куратором проекта или специалистом предметной области, чтобы узнать, насколько может в будущем вырасти пользовательская база, объем данных и другие параметры. В процессе этого обсуждения могут быть полезными следующие вопросы:

- Какова оценка числа общего числа и числа одновременных пользователей, которое должна обслуживать система через несколько месяцев, кварталов или лет?
- Не могли бы вы описать, как и почему в будущем могут вырасти потребности в мощностях хранения данных?
- Какие минимально приемлемые критерии производительности должны быть удовлетворены независимо от числа пользователей?
- Каковы планы будущего роста в плане числа серверов, центров обработки данных или числа установленных экземпляров системы?

Нет, пожалуйста, не уходите!

«Кибер-понедельник» — день, наступающий после Черной пятницы, которая дает старт сезону распродаж в Соединенных Штатах, между Днем благодарения и Рождеством. Он стал традиционным днем, когда люди активно покупают подарки в интернет-магазинах. Когда эта традиция началась в начале 2000-х годов, многие веб-сайты электронной коммерции оказались не готовы к обслуживанию пикового трафика и транзакций клиентов, ищущих лучших цен. Серверы «падали», пароли не принимались, а процесс покупки сильно затягивался вплоть до полной невозможности ее совершить. Многие покупатели покидали интернет-магазины, отправлялись в поиске подарков в другое место и, как правило, больше не возвращались. Киберпреступники воспользовались этим, создав похожие на настоящие подставные сайты, на которые перенаправлялись покупатели, и использовав эти сайты для сбора личной информации покупателей.

Эти проблемы иллюстрируют взаимосвязанный клубок неудовлетворенных требований к качеству ПО. Из-за недостаточной масштабируемости при большом наплыве посетителей системы начали испытывать проблемы с надежностью, а это привело к снижению доступности. Качество ПО напрямую влияет на доходность компании.

Проверяемость

Проверяемость (verifiability) еще называют тестируемостью, и она показывает легкость, с которой программные компоненты или интегрированный продукт можно проверить на предмет реализации функций системы так, как ожидалось. Проектирование с расчетом на проверяемость критически важно для продукта, в котором используются сложные алгоритмы и логика или имеются тонкие функциональные взаимосвязи. Тестируемость также важна в том случае, если продукт необходимо часто модифицировать, поскольку предполагается подвергать его частому регрессивному тестированию, чтобы выяснить, не ухудшают ли внесенные изменения существующую функциональность. Системы с высокой проверяемостью эффективно поддаются тестированию. Проектирование с расчетом на проверяемость означает простоту перевода ПО в требуемое состояние, предшествующее тестированию, предоставление необходимых тестовых данных и наблюдение за результатом тестирования. Далее приводится несколько примеров требований к проверяемости:

VER-1. Конфигурация среды разработки должна быть идентичной конфигурации среды тестирования, чтобы предотвратить невоспроизводимость дефектов, выявляемых в тесте.

VER-2. У тестировщика должна быть возможность конфигурировать, какие результаты будут записываться во время тестирования.

VER-3. У разработчика должна быть возможность настраивать вычислительный модуль на отображение промежуточных результатов любой заданной группы алгоритмов для целей отладки.

Поскольку я и команда разработчиков твердо знали, что нам придется тестировать Graphics Engine много раз в период его неоднократных усовершенствований, мы включили следующее указание, касающееся тестируемости ПО:

VER-4. Максимальная цикломатическая сложность модуля не должна превышать 20.

Цикломатическая сложность (cyclomatic complexity) — это количество логических ответвлений в модуле исходного кода. Чем больше ответвлений и циклов в модуле, тем сложнее его тестировать, понимать и поддерживать. Конечно, проект не будет провален, если значение цикломатической сложности одного из модулей достигнет 24, однако наше указание определило для разработчиков уровень качества, к которому следует стремиться.

Определение требований к проверяемости может оказаться непростой задачей. Формулируйте вопросы примерно так:

- Как можно подтвердить, что определенные вычисления дают ожидаемые результаты?
- Есть ли какие-то части системы, не обеспечивающие детерминированных результатов, из-за чего сложно определить, правильно ли они работают?
- Возможно ли создать набор тестовых данных, которые с высокой вероятностью позволять выявить какие-либо ошибки в требованиях или их реализации?
- Какие стандартные отчеты или другие результаты можно использовать для проверки того, что система возвращает правильные результаты?

Определение требований по качеству с помощью языка Planguage

Вам не удастся оценить, отвечает ли продукт неточно сформулированным требованиям к качеству или нет. Непроверяемые требования к качеству ничем не лучше непроверяемых функциональных требований. Упрощенные задачи, связанные с качеством и производительностью, могут оказаться невыполнимыми. Меньшее секунды время отклика на запрос базы данных может замечательно работать при несложном поиске в локальной БД, но абсолютно невыполнимо при соединении шести связанных таблиц на географически разнесенных серверах.

Для решения проблемы неоднозначных и неполных нефункциональных требований консультант Том Гилб (Tom Gilb, 1997; 2005) разработал Planguage, язык с большим набором ключевых слов, который позволяет точно формулировать атрибуты качества и другие задачи проекта (Simmons, 2001). Далее показано, как выразить требование к производительности с помощью всего лишь нескольких из множества ключевых слов Planguage. Будучи выраженным в традиционной форме, это требование звучит так: «Как минимум 95% времени системе должно требоваться не более 8 секунд на отображение любого из предопределенных бухгалтерских отчетов».

- **TAG** Performance.Report.ResponseTime
- **AMBITION** Быстрый отклик на запросы построения бухгалтерских отчетов на базовой пользовательской платформе.
- **SCALE** Время (в секундах) между нажатием клавиши Enter или щелчком ОК для отправки запроса и началом отображения отчета.
- **METER** Тестирование с использованием секундомера, выполненное на 30 тестовых запросах, представляющих определенный операционный профиль бухгалтера.
- **GOAL** Не более 8 в 95% отчетов. ← Начальник отдела.

- **STRETCH** Не более 2 секунд для предопределенных отчетов, 5 секунд для всех отчетов.
- **WISH** Не более 1,5 секунд для всех отчетов.
- **base user platform DEFINED** Процессор Intel Core 2 Quad, оперативная память 8 ГБ, под управлением ОС Microsoft Windows 8, с установленным пакетом QueryGen версии 3.3, однопользовательский компьютер, свободно как минимум 50% оперативной памяти и 70% процессорных ресурсов, сетевое подключение со скоростью 30 Мбит/с.

Каждому требованию назначается уникальный тег (tag), или метка, иерархической системы именования, описанной в главе 10. AMBITION (цель) устанавливает цель или задачу для системы, которая порождает данное требование. SCALE определяет единицы измерения, а METER (счетчик) описывает точно, как выполнить измерения. Все заинтересованные лица должны одинаково хорошо понимать, что означает «производительность». Предположим, пользователь проще воспринимает систему измерений, которая основана на времени от нажатия клавиши Enter до вывода всего отчета, чем до начала отображения отчета, как указано в примере. Разработчик может заявить, что требование удовлетворено, а пользователь будет утверждать обратное. Точно выраженные требования к качеству и системе измерений помогут предотвратить такого рода недоразумения.

Одно из преимуществ Planguage — возможность указать несколько желаемых количественных величин для измерения качества. Критерий MUST (обязательство) определяет наименьший приемлемый уровень. Требование не удовлетворено до тех пор, пока не будут удовлетворены все условия GOAL, поэтому нужно убедиться, что эти условия обоснованы в бизнес-терминах. Можно указать требование GOAL другим способом — определить условие FAIL (неудача) (еще одно ключевое слов языка Planguage), например: «Более 8 секунд ожидания для более 2% всех отчетов». Величина STRETCH описывает желательную производительность, а WISH — идеальный результат. Кроме того, стоит показать источник требуемой производительности. Стрелка (←) после условия GOAL показывает, что оно поступило от Начальника отдела. Для облегчения чтения любые специализированные термины, используемые в операторах языке Planguage, заданы как DEFINED (то есть определенными). Этот пример предоставляет определение того, что называется базовой пользовательской платформой (Base User Platform), на которой будут выполняться тесты.

Planguage включает много дополнительных ключевых слов, с помощью которых удается добиться более гибкой и точной спецификации требований к атрибутам качества и даже бизнес-целям. Указав несколько уровней для ожидаемого результата, вы получите более широкое представление о требованиях к качеству, чем с помощью простых «черных–белых» конструкций типа «да–нет». Недостаток Planguage заключается в том, что получаемые в результате требования намного объемнее, чем простые формулировки тре-

бований к качеству. Однако богатство предоставляемой информации перевешивают это неудобство. Даже если вы не используете весь формальный аппарат Planguage для описания требований к качеству, применение ключевых слов для более четкого понимания, что пользователи понимают под «быстро» позволит получить более точно и согласованно понять ожидания.

Компромиссы для атрибутов

Для определенных комбинаций атрибутов компромиссы неизбежны. Пользователи и разработчики должны решить, какие атрибуты важнее других и при принятии решений соблюдать эти приоритеты. В таком анализе может помочь прием, описанный ранее в разделе «Этап 3. Определение приоритетов атрибутов». На рис. 14-2 показаны типичные взаимосвязи атрибутов качества, перечисленных в табл. 14-1, однако следует знать, что возможны исключения (Charette, 1990; Glass, 1992; IEEE, 1998). Знак «+» в ячейке означает, что увеличение величины атрибута в соответствующей строке позитивно влияет на атрибут в соответствующем столбце. Например, при повышении легкости перемещения программного компонента повышается гибкость ПО, упрощается подключение к компонентам другого ПО и возможность повторного использования и тестирования.

	Доступность	Эффективность	Удобство установки	Целостность	Совместимость	Возможность модификации	Производительность	Переносимость	Надежность	Возможность повторного использования	Устойчивость	Защита	Масштабируемость	Безопасность	Удобство использования	Проверяемость и тестируемость
Доступность								+		+						
Эффективность	+				–	–	+	–			–		+		–	
Удобство установки	+							+						+		
Целостность			–		–		–			–		+		+	–	–
Совместимость	+		–	–			–	+	+		+	–		–		
Возможность модификации	+		–				–		+	+			+			+
Производительность		+			–	–		–			–		–		–	
Переносимость		–			+	–	–			+				–	–	+
Надежность	+	–		+		+	–				+	+		+	+	+
Возможность повторного использования		–		–	+	+	–	+						–		+
Устойчивость	+	–	+	+	+		–		+			+	+	+	+	
Защита		–		+	+		–				+			+	–	–
Масштабируемость	+	+		+			+	+	+		+					
Безопасность	+			+	+		–	–	+		+	+			–	–
Удобство использования		–	+				–	–	+		+	+				–
Проверяемость и тестируемость	+		+	+		+			+	+	+	+		+	+	

Рис. 14-2. Позитивные и негативные взаимосвязи некоторых атрибутов качества

Знак «–» в ячейке означает, что увеличение величины атрибута в этой строке негативно влияет на атрибут в соответствующем столбце. Пустая ячей-

ка означает, что атрибут в этом строке оказывает незначительное влияние на атрибут в столбце. Производительность и эффективность оказывают негативное влияние на многие другие атрибуты. Если вы напишете самую компактную и быструю программу, используя хитрости кодирования и учитывая некоторые побочные эффекты при выполнении, вероятнее всего эту программу будет сложно поддерживать и исправлять, а также переносить на другие платформы, если вы «заточили» программу под конкретную операционную среду. Аналогично, системы, которые были оптимизированы для удобства использования или спроектированы как гибкие, пригодные для многократного использования и способные взаимодействовать с другими программными компонентами или компонентами оборудования, часто грешат проблемами производительности. При создании чертежей средствами универсального компонента Graphics Engine, о котором говорилось ранее в этой главе, производительность понизилась по сравнению с приложением, где использовался нестандартный код для обработки графики. Необходимо найти баланс между возможным ухудшением производительности и ожидаемой выгодой от предлагаемого решения, чтобы убедиться, что компромисс, на который вы идете, разумен.

Матрица на рис. 14-2 не симметрична, потому что влияние при увеличении атрибута A на атрибут B не обязательно такое же, как влияние, которое оказывает увеличение атрибута B на атрибут A. Так, из рис. 14-2 видно, что модификация системы с целью повышения производительности не оказывает влияния на безопасность. Но повышение безопасности вероятнее всего весьма отрицательно отразится на производительности, поскольку системе придется выполнять много дополнительных операций: аутентификацию пользователей, шифрование и проверку на наличие вирусов.

Чтобы оптимизировать баланс характеристик продукта, в процессе выявления требований следует определить и задокументировать важнейшие атрибуты качества и расставить для них приоритеты. При определении важных для проекта атрибутов качества используйте рис. 14-2 для того, чтобы не поставить перед собой несовместимые цели. Как это может быть, показано ниже.

- Не пытайтесь максимизировать удобство использования, если ПО должно работать на многих платформах при минимальных изменениях (мобильность). Разные платформы и операционные системы налагают различные ограничения и другие характеристики удобства использования.
- Нелегко полностью проверить требования к целостности систем с высоким уровнем безопасности. Повторно используемые универсальные компоненты могут снизить надежность механизмов безопасности.
- Крайне надежный код может оказаться менее производительным из-за постоянного выполнения им проверок данных и поиска ошибок.

Как правило, слишком ограничивая возможности системы или определяя конфликтующие требования, вы тем самым усложняете задачу разработчи-

кам — им гораздо труднее удовлетворить все требования заказчика.

Реализация требований к атрибутам качества

Дизайнеры и программисты должны определить наилучший способ удовлетворения требования для каждого атрибута качества. Хотя атрибуты качества относятся к нефункциональным требованиям, они могут служить источником производных функциональных требований, определять направления дизайна или технические сведения, которые позволяют реализовать продукт желаемого качества. В табл. 14-5 перечислены некоторые категории технической информации, которые могут потребоваться по причине наличия атрибутов качества. Например, в медицинское устройство со строгими требованиями к доступности может входить источник резервного питания (архитектура) и функциональные требования, оговаривающие визуальное или звуковое оповещения, когда продукт работает на резервном питании. Этот переход от внешних или внутренних требований к качеству к соответствующей технической информации является частью процесса разработки требований и высокоуровневого дизайна.

Табл. 14-5. Преобразование требований к качеству в техническую документацию

Атрибуты качества	Возможная категория технической информации
Удобство установки, целостность, совместимость, надежность, устойчивость, защита, безопасность, удобство использования и проверяемость	Функциональное требование
Доступность, эффективность, возможность модификации, производительность, надежность и масштабируемость	Системная архитектура
Совместимость, безопасность, удобство использования	Ограничение дизайна
Эффективность, возможность модификации, переносимость, надежность, возможность повторного использования, масштабируемость, проверяемость, удобство использования	Рекомендации по дизайну
Переносимость	Ограничение реализации

Бизнес-аналитики, не обладающие достаточным опытом разработки, могут не полностью понимать технических последствий требований к качеству. Поэтому бизнес-аналитик должен привлекать правильных заинтересованных лиц, понимающих возможные последствия, и получать от них нужную информацию. Возьмем для примера масштабируемость, на которую сильное влияние может оказать выбор архитектуры и дизайна. Требования к масштабируемости могут заставить разработчика предусмотреть запас для произ-

водительности (дисковое пространство, процессорные ресурсы, пропускная способность), чтобы при росте в будущем это не снизило производительность до недопустимых величин. Ожидания масштабируемости могут влиять на решения разработчиков в области оборудования и операционной среды. Вот почему важно выявлять и документировать требования к масштабируемост на ранних этапах, чтобы разработчики могли обеспечить приемлемую производительность при запланированном росте продукта. Это еще одна причина важности привлечения разработчиков и рецензентов на ранних этапах сбора требований.

Ограничения

Ограничение накладывает границы на доступный разработчику выбор дизайна и реализации. Ограничения могут накладываться внешними заинтересованными лицами, другими системами, которые взаимодействуют с создаваемой или обслуживаемой вами или другими действиями жизненного цикла вашей системы, такими как миграция и обслуживание. Другие ограничения проистекают из существующих соглашений, решений руководства и технических решений (ISO/IEC/IEEE, 2011). Источники ограничения могут быть такого рода:

- определенные технологии, средства, языки программирования и базы данных, которые следует использовать или избегать;
- ограничения, налагаемые операционной средой или платформой продукта, например типы и версии установленных веб-браузеров или используемых операционных систем;
- обязательные соглашения или стандарты разработки (например, если обслуживать ПО будут клиенты, то они должны указать нотацию дизайна и стандарты программирования, которые субподрядчик обязан соблюдать);
- обратная совместимость с ранее выпущенными продуктами и возможности совместимости с будущими версиями, например понимание, какая версия программы использовалась для создания конкретного файла;
- ограничения или требования, налагаемые нормативными документами и другими бизнес-правилами;
- ограничения, связанные с оборудованием, например требования к срокам, ограничения памяти или процессора, размер, вес, материалы или затраты;
- физические ограничения, обусловленные операционной средой или характеристиками или ограничениями пользователей;
- соглашения, связанные с пользовательским интерфейсом существующего продукта, которые необходимо соблюдать при улучшении существующего продукта;
- интерфейсы с другими существующими системами, такие как форматы данных и протоколы связи;

- ограничения, связанные с размером экрана, например при работе на планшете или смартфоне;
- стандартный формат обмена данными, например XML или RosettaNet для электронного бизнеса.

Ограничения такого типа часто накладываются внешними источниками и должны соблюдаться. Но ограничение могут накладываться непреднамеренно. Очень часто пользователи представляют «требования» как идеи решений, которые описывают, как они представляют себе решение задачи. Бизнес-аналитик должен обнаруживать, когда требование включает предложение решения и отделить базовую потребность от ограничения, накладываемого этим решением. Возможно, предлагаемое пользователем решение — идеальный способ решения проблемы, тогда ограничение вполне правомерно. Но чаще реальная потребность скрыта, и бизнес-аналитик должен поработать с пользователем над четким выражением мыслей пользователя, которые привели его к этому решению. Многократное повторение вопроса «почему?» обычно позволяет выяснить реальное требование.

Некоторые говорят, что атрибуты качества *и есть* ограничения. Мы предпочитаем думать, что определенные требования к качеству служат источником некоторых ограничений дизайна и реализации. Как показано в табл. 14-5, требования к совместимости и удобству использования являются возможными источниками ограничений дизайна. Переносимость часто накладывает ограничения реализации, которые призваны обеспечивать легкость перехода между платформами или операционными средами. Например, одни компиляторы определяют размер типа данных *integer* в 32 бита, а другие — 64 бита. Чтобы выполнить требования к переносимости, программисту надо в символической форме определить тип данных WORD как 32-разрядное целое без знака и использовать тип данных WORD вместо целочисленного типа данных, принятого в компиляторе по умолчанию. Таким образом гарантируется, что все компиляторы будут одинаково обращаться к элементам данных типа WORD, что сделает работу системы предсказуемой в различных операционных средах.

Далее приводится несколько примеров ограничений. Видно, как они ограничивают возможности архитектора, дизайнера и разработчика.

CON-1. Пользователь щелчком в верхней части списка проектов изменяет порядок сортировки. [*конкретный элемент управления на пользовательском интерфейсе, примененный как ограничение дизайна на функциональное требование*]

CON-2. Для реализации продукта может использоваться ПО с открытым кодом, доступное в рамках лцензии GNU. [*ограничение реализации*]

CON-3. Приложение должно использовать Microsoft .NET Framework 4.5. [*архитектурное ограничение*]

CON-4. Банкомат содержит только 20-долларовые банкроты. [*физическое ограничение*]

CON-5. Интернет-платежи могут выполняться только через PayPal. [*ограничение дизайна*]

CON-6. Все используемые в приложении текстовые данные должны храниться в виде XML-файлов. [*ограничение данных*]

Заметьте, что некоторые из этих ограничений существуют для выполнения некоторых, возможно невысказанных ожиданий. Для каждого ограничения спросите, почему оно накладывается для достижения базовых требований к качеству. Почему в ограничении CON-2 должно использоваться ПО с открытым кодом? Возможно, причина — желание повысить возможность модификации, поэтому требование ведет к ограничению. Почему ограничение CON-3 требует использовать вполне определенную версию .NET? Возможно, причина заключается в неявном требовании к переносимости или надежности. Помните: ограничение — это предполагаемое решение, а дополнительные вопросы «почему?» позволяют добраться до сути — до требования, для удовлетворения которого предназначено это решение.

Атрибуты качества в проектах гибкой разработки

Встраивание требуемых характеристик качества может оказаться сложным и дорогим, если это делается на поздних этапах разработки или после поставки продукта. Поэтому в проектах гибкой (agile) разработки, в которых разработка требований и поставка функциональности происходит небольшими порциями, нужно определять важные атрибуты качества и ограничения на ранних этапах проекта. Это позволяет разработчикам принимать соответствующие решения по архитектуре и дизайну, которые будут положены в основу требуемых характеристик качества. Нефункциональным требованиям приоритеты должны назначаться параллельно с пользовательскими историями; нельзя откладывать реализацию этих требований на следующую итерацию.

Атрибуты качества возможно определять в форме историй:

Как техник отдела поддержки я хочу, чтобы база знаний реагировала на запросы в течение 5 секунд, чтобы клиент не разочаровывался и не отключался.

Однако требования к качеству не реализуются так же дискретно, как пользовательские истории. Они могут охватывать много историй и итераций. Кроме того эти требования не всегда возможно разбить на более мелкие части, которые можно было бы реализовывать на протяжении многих итераций, как это происходит с пользовательскими историями.

Обдумывая последствия реализации конкретных пользовательских историй, разработчики должны держать в голове нефункциональные требования. По мере добавления большего объема функциональности в течение серии итераций эффективность, а, значит, и производительность, системы может упасть. Определите цели производительности и начинайте тестирование

производительности на ранних итерациях, чтобы вовремя узнать о проблемах и как можно раньше их устранить.

Как вы видели в табл. 14-5, некоторые атрибуты качества являются источником производной функциональности. В проекте гибкой разработки требования к качеству могут добавить новые элементы в резерв (backlog) проекта. Посмотрите на следующее требование к безопасности:

Как владелец учетной записи я хочу предотвратить доступ неуполномоченных пользователей к моей учетной записи, чтобы я не потерял деньги.

Это требование может заставить владельца продукта или бизнес-аналитика проекта создать много производных пользовательских историй, описывающих относящуюся к безопасности функциональность. Эти истории могут добавляться в резерв проекта и стандартным образом планироваться для реализации в конкретных итерациях. Понимание этих требований с самого начала гарантирует, что команда вовремя реализует требования к безопасности.

Как и для пользовательских историй, можно создавать тесты для атрибутов качества. Это один из способов количественной оценки атрибутов качества. Если цель производительности сформулирована просто: «База знаний должна быстро возвращать результаты поиска», невозможно написать тесты, позволяющие удостовериться, что система делает это «быстро». Вот пример более правильного приемочного теста:

Получение результата при поиске по ключевым словам в базе знаний должно занимать не более 5 секунд, но желательно менее 3 секунд.

Приемочные тесты в такой форме могут содержать несколько уровней удовлетворения требований, во многом как делают такие ключевые слова, как GOAL, STRETCH и WISH в обсужденном ранее в этой главе языке Planguage. Ключевые слова SCALE и METER языка Planguage можно использовать для более точного определения, что конкретно означает «вернуть результат» и как выполнить тест и оценить результаты.

Удовлетворение нефункциональных требований — один из критериев, по которым определяется, завершена ли итерация. Часто существует диапазон допустимой производительности, при этом одни результаты считаются предпочтительнее других. Как и в других методиках разработки ПО, удовлетворение требований по качеству может стать тем, что отличает провальный результат проекта гибкой разработки от крайне успешного продукта, которым пользователи в высшей степени довольны.

Что дальше?

- Определите несколько атрибутов качества для пользователей из табл. 14-1, которые могут быть важными для пользователей вашего текущего проекта. Сформулируйте несколько вопросов о каждом атрибуте, которые помогут пользователям сформулировать их ожидания. Основываясь на ответах пользователей, письменно сформулируйте одну или несколько задач для каждого важного атрибута.

- Изучите несколько задокументированных требований к качеству вашего проекта на предмет их проверяемости. Если этот показатель невысок, перепишите их так, чтобы можно было оценить достижение ожидаемых результатов по качеству в проекте.
- Вернитесь к разделу «Изучение атрибутов качества» этой главы и попытайтесь применить описанный в нем табличный подход к размещению по приоритету важных атрибутов качества. Есть ли компромиссы между атрибутами в вашем проекте в соответствии с анализом приоритетов?
- Перепишите несколько примеров атрибутов качества из этой главы с помощью Planguage, при необходимости для наглядности делая некоторые допущения. Удается ли вам с помощью Planguage выразить эти требования к качеству более точно и недвусмысленно?
- Изучите ожидания пользователей, касающиеся качества системы, на предмет возможных противоречий и устраните их. Привилегированные классы пользователей должны иметь наибольшее влияние при принятии компромиссов.
- Отследите, как атрибуты качества влияют на функциональные требования, ограничения дизайна и реализации или выбор архитектуры и дизайна.

Глава 15

Прототипы как средство снижения риска

«Шэрон, сегодня я хотел бы поговорить с вами о требованиях покупателей в отделе закупок к новой Chemical Tracking System, — начал бизнес-аналитик Лори. — Вы не могли бы мне сказать, что система должна делать?»

«Ничего себе, я даже не знаю, как к этому подступиться, — ответила Шэрон озадаченно. — Я не могу описать, что мне нужно, но я буду знать, когда увижу это».

«Узнаю, когда увижу» (IKIWISI — «I'll know it when I see it»), — фраза, от которой в жилах аналитиков требований стынет кровь. Сразу в воображении возникает картина: команда разработчиков воплощает свои лучшие идеи в разработке ПО, увидев которое пользователи говорят: «Нет, не то, попробуйте еще раз». Действительно, представить будущую программную систему и ясно изложить требования к ней — нелегкая задача. Многим людям сложно описать свои потребности, не имея перед собой чего-то осязаемого, без образца, да и критиковать гораздо легче, чем создавать.

Создание прототипов ПО — пробный шаг в мир решений. Оно делает требования более реальными, приближает варианты использования к реальности и закрывает пробелы в вашем понимании требований. Прототипы предоставляют пользователям экспериментальную модель или первоначальный срез новой системы, стимулируя их мышление и катализируя обсуждение требований. Обсуждение прототипов на ранних стадиях процесса разработки помогает заинтересованным в проекте лицам прийти к общему пониманию требований к системе, что уменьшает риск недовольства заказчиков.

Даже при применении методов разработки требований из предыдущих глав отдельные их части могут быть неопределенными или неясными как для клиентов, так и для разработчиков, или для тех, и других. Если вы не исправите эти проблемы, разница в ожиданиях между видением продукта пользователями и пониманием разработчиков гарантирована. Создание прототипов — мощный способ привлечения очень важных контактов от клиента, которые позволят если не закрыть, то значительно сузить разрыв ожиданий. Трудно точно представить поведение нового ПО при чтении текстовой документации или изучении аналитических моделей. Пользователи гораздо охотнее согласятся поэкспериментировать с прототипом (что интересно),

чем читать спецификацию требований к программному обеспечению (что скучно). Услышав от пользователей «Узнаю, когда увижу», подумайте, как вы можете помочь им наглядно представить свои нужды или лучше понять свои мысли (Boehm, 2000). Прототипы также ценный инструмент проверки требований. Бизнес-аналитик может попросить пользователей поработать с прототипами, чтобы понять, удовлетворит созданный на основе этих прототипов продукт их потребности.

Слово *прототип* (prototype) имеет множество значений, поэтому ожидания участников процесса создания прототипов могут весьма различаться. Прототип самолета действительно летает — ведь это первый вариант настоящего самолета. Напротив, прототип ПО — это только часть или образец реальной системы — он может в принципе не делать ничего полезного. Прототипы ПО могут быть действующими моделями или статичными образцами; детализированным изображением экрана или его эскизами; наглядной демонстрацией или примерами реальной функциональности; имитацией или подражанием (Stevens и др., 1998; Constantine и Lockwood, 1999).

В этой главе описаны различные виды прототипов, их использование во время разработки требований к ПО и способы эффективного внедрения прототипов в процесс разработки ПО. Здесь также содержатся советы, как использовать их в процессе разработки требований, а также как сделать создание прототипов частью вашего процесса разработки ПО.

Что такое прототип и для чего он нужен

Прототип ПО — это частичное, возможное и предварительное воплощение предлагаемого нового продукта. Прототипы служат трем основным целям и эти цели должны быть ясны с самого начала:

- **Прояснение, формулировка и утверждение требований** Используемый в качестве инструмента формулировки требований, прототип позволяет достичь соглашения, обнаружить ошибки и оценить точность и качество требований. Оценка прототипа пользователями указывает на ошибки в формулировке требований, которые можно исправить без больших затрат до создания реального продукта. Это особенно полезно в тех частях системы, которые не очень понятны или особенно рискованны или сложны;
- **Исследование альтернативных решений** Прототип, как инструмент конструирования, позволяет заинтересованным в проекте лицам исследовать различные варианты реализации взаимодействия пользователей, наглядно представить конечный продукт, оптимизировать удобство работы и оценить возможные технические приемы. Прототипы позволяют на рабочих образцах показать на примере рабочих конструкций, насколько осуществимы требования. Они полезны для того, чтобы убедиться, что разработчики правильно понимают требования до начала работы над реальным решением;

- **Создание подмножества, которое в конечном итоге превратится в конечный продукт** Используемый в качестве инструмента разработки, прототип — не что иное, как функциональная реализация подмножества продукта, которую можно превратить в готовый продукт в процессе последовательной цепочки небольших циклов разработки. Это безопасный подход, только если прототип тщательно с самого начала спроектирован с прицелом на конечный выпуск.

Основная цель создания прототипа — устранение неясностей на ранних стадиях процесса разработки. Для этого не нужен прототип всего продукта. Надо сосредоточиться на высокорисковых областях и известных неясностях и именно исходя из них следует решать, для каких частей системы необходим прототип и что вы надеетесь выяснить, оценивая его. Прототип полезен для выявления и устранения двусмысленных и неполных утверждений в требованиях. Пользователи, менеджеры и другие заинтересованные лица проекта считают, что прототип дает им понимание некой конкретики, пока реальный продукт документируется и разрабатывается. Для каждого создаваемого прототипа обязательно указывайте — и доводите до заинтересованных лиц, почему он создается, что вы хотите узнать с его помощью и что вы будете делать с прототипом после его оценки.

Во избежание недопонимания важно добавить какие-то описательные слова перед словом «прототип», чтобы участники проекта понимали, почему и когда вы можете создавать прототип того тот или иного типа. В этой главе описываются три классы атрибутов прототипа, у каждого из которых есть две альтернативы:

- **назначение** Модель ориентирована на пользовательский интерфейс, а экспериментальный образец — на верность технического решения;
- **использование в будущем** Одноразовый прототип отбрасывается после получения отзывов, эволюционный — превращается в готовый продукт в процессе серии итераций;
- **форма** Бумажный прототип представляет собой простой набросок на бумаге, доске или в программе рисования, а электронный прототип состоит из рабочего ПО только одной части решения.

Каждый создаваемый прототип обладает определенным сочетанием этих свойств. Например, можно создать одноразовую бумажную модель с простыми рисунками возможных экранов. Или можно построить эволюционный экспериментальный образец — рабочее ПО, которое демонстрирует требуемые технические характеристики, которые потом можно преобразовать в готовый продукт. Но некоторые сочетания бессмысленны. Например, нельзя создать эволюционный бумажный экспериментальный образец.

Модели и экспериментальные образцы

Когда люди говорят «прототип ПО», они обычно имеют в виду *модель* (mock-up) предполагаемого интерфейса пользователя. Его также именуют *горизонтальным прототипом* (horizontal prototype). Такой прототип ориентирован на какую-то часть пользовательского интерфейса и не затрагивает уровни архитектуры или подробности функциональности. Прототип этого типа позволяет пользователям исследовать поведение предполагаемой системы в тех или иных ситуациях для уточнения требований. Модель также позволяет пользователям выяснить, смогут ли они с помощью системы, основанной на прототипе, выполнять свою работу.

Модель, подобно кинофильму, создает видимость функциональности, не обеспечивая ее реального воплощения. Она показывает внешний вид экранов пользовательского интерфейса и позволяет осуществлять частичную навигацию между ними, но не содержит никакой или почти никакой реальной функциональности. Это как типичный вестерн: ковбой входит в салун, а затем выходит из конюшни, но он не заказывает выпивку и не видит лошадей, потому что за фасадами декораций ничего нет.

Модели могут демонстрировать функциональные возможности, которые будут доступны пользователю, внешний вид и поведение пользовательского интерфейса (цвета, планировку, графику, элементы управления) и структуру доступа к информации (структуру навигации). Навигация между объектами интерфейса возможна, но вместо некоторых элементов пользователь увидит лишь краткое сообщение о том, что будет здесь находиться. Информация, появляющаяся в ответ на запрос к базе данных, может быть имитированной или неизменяющейся, а содержание отчетов — жестко закодированным. Создавая модели, старайтесь использовать реальные данные в образцах отчетов, диаграмм и таблиц — это увеличит достоверность прототипа как модели реальной системы, но обязательно позаботьтесь, чтобы рецензенты прототипа понимали, что все формы и результаты не настоящие, а имитация.

Модель не выполняет полезной работы, хотя иногда и кажется, что да. Зачастую имитации вполне достаточно, чтобы пользователи могли решить, нет ли каких-либо упущений, неверных или ненужных функций. Некоторые прототипы выражают концепцию реализации конкретных вариантов использования, как ее видят разработчики. Оценивая прототип, пользователи могут указать на альтернативные направления вариантов использования, пропущенные шаги взаимодействия, дополнительные исключения, пропущенные выходные условия и уместные бизнес-правила.

Работая с одноразовой моделью, пользователь должен концентрироваться на общих требованиях и вопросах последовательности выполняемых действий, не отвлекаясь на особенности внешнего вида элементов экрана (Constantine, 1998). На этой стадии не надо беспокоиться о точном расположении элементов экрана, шрифтах, цветах и графике. Время детализации интерфейса пользователя наступит после того, как прояснятся требования к системе и будет

определена общая структура интерфейса. Что касается эволюционных моделей, то такие операции продвигают пользовательский интерфейс ближе к реализации и выпуску настоящего интерфейса.

Экспериментальный образец (proof of concept), также называемый *вертикальным прототипом* (vertical prototype), воплощает срез функциональности приложения от интерфейса пользователя через все уровни технических сервисов. Экспериментальный образец действует как настоящая система, поскольку затрагивает все уровни ее реализации. Разрабатывайте экспериментальный образец всякий раз, когда сомневаетесь в осуществимости и стабильности предполагаемого подхода к архитектуре системы или когда хотите оптимизировать алгоритмы, оценить предлагаемую схему базы данных или проверить критически важные временны*е* требования. Чтобы получить осмысленные результаты, экспериментальные образцы создают, используя средства разработки в среде, аналогичной среде разработки. Экспериментальные образцы также полезны для сбора информации для совершенствования способности команды оценивать усилия по реализации определенной пользовательской истории или блока функциональности. В проектах гибкой разработки (agile) экспериментальный образец иногда называют *спайком* (spike).

Однажды я работал с командой, которая занималась реализацией необычной клиент-серверной архитектуры для стратегии перехода от мэйнфрейм-системы к программной среде, основанной на серверах и рабочих станциях, работающих под UNIX и объединенных в сеть (Thompson и Wiegers, 1995). Экспериментальный образец, в котором была реализована небольшая часть клиента пользовательского интерфейса (на мэйнфрейме) и соответствующая часть функциональности сервера (на рабочей станции под UNIX), позволил нам оценить компоненты связи, производительность и надежность предлагаемой архитектуры. Эксперимент прошел успешно, как и построение решения на основе этой архитектуры.

Одноразовые и эволюционные прототипы

Прежде чем создавать прототип, примите четкое и ясное решение, прекратите ли вы работать с ним после оценки или сделаете его частью выпускаемого продукта. Создайте *одноразовый прототип* (throwaway prototype), чтобы ответить на вопросы, разрешить неясности и улучшить требования к ПО (Davis, 1993). Если вы решили, что после выполнения задачи прекратите работу с прототипом, стройте его как можно более быстро и дешево. Чем больше усилий вы вложите в прототип, тем труднее будет участникам проекта отказаться от него и тем меньше времени вам останется для построения реального продукта.

Не стоит уничтожать прототип, если вы видите возможность его повторного применения в будущем. Тем не менее, он не должен стать частью окончательного продукта. Поэтому его можно называть *невыпускаемым прототипом* (nonreleasable prototype).

Создавая одноразовый прототип, разработчики пренебрегают большинством известных им методов конструирования качественного ПО. В одноразовом прототипе предпочтение отдается быстроте воплощения и модификации, а не устойчивости, надежности, производительности и долговременному удобству сопровождения. Поэтому внимательно следите, чтобы низкокачественный код одноразового прототипа не попал в окончательный продукт. В противном случае пользователи и персонал технического обслуживания будут страдать от последствий этого в течение всего срока эксплуатации продукта.

Одноразовый прототип наиболее уместен, когда команда разработчиков сталкивается с неизвестностью, двусмысленностью, неполнотой или неопределенностью в требованиях к ПО или когда возникают сложности с наглядным представлением системы на основе одних требований. Разрешение этих вопросов уменьшает риск продолжения разработки. Прототип, помогающий пользователям и разработчикам визуально представить реализацию требований, может выявить пробелы в документации, а также решить, годятся ли требования для создания продукта, поддерживающего необходимые бизнес-процессы.

> **Внимание!** Не делайте одноразовый прототип сложнее, чем это необходимо для целей его создания. Не поддавайтесь искушению — или давлению пользователей — добавить в прототип больше функций.

Каркас (wireframe) — определенная форма одноразового прототипа, которая используется для дизайна нестандартных пользовательских интерфейсов или веб-сайтов. Каркасы можно применять для лучшего понимания трех характеристик веб-сайта:

- концептуальных требований;
- архитектуры информации или дизайна навигации;
- подробного дизайна страниц высокого разрешения.

Наброски страниц при изучении концептуальных требований в первом типе каркаса не обязательно должны быть похожими на окончательные страницы. Этот каркас применяется для работы с пользователями, чтобы понять, какие действия, они могут захотеть выполнять на экране. Для этой цели хорошо подходят бумажные прототипы, о которых подробнее говорится ниже в этой главе. Второй тип каркаса может вообще не касаться дизайна страниц. Описанная в главе 12 модель анализа «карта диалоговых окон» — отличный инструмент для изучения и пошагового прохода навигации по страницам веб-сайта. Третий тип каркаса углубляется в детали внешнего вида окончательных страниц.

В отличие от одноразового прототипа, *эволюционный прототип* (evolutionary prototype) представляет собой прочный архитектурный «фундамент» для постепенного создания окончательного продукта — по мере прояснения требований (McConnell, 1996). Гибкая разработка (agile) — пример использования эволюционного прототипа. Команды гибкой разработки разрабатывают продукт в процессе серии итераций, используя отзывы пользователей на ранних итерациях для корректировки направления будущих циклов разработки. Это суть применения эволюционных прототипов.

В отличие от одноразовых прототипов, создаваемых «быстро и начерно», для построения эволюционного прототипа необходимо с самого начала использовать отличный и качественный код. Поэтому на конструирование эволюционного прототипа требуется больше времени, чем на одноразовый прототип, моделирующий те же свойства системы. Эволюционный прототип следует создавать в расчете на легкий рост и частое расширение, поэтому разработчики должны уделять большое внимание архитектуре и принципам проектирования. При создании такого прототипа ни в коем случае не стоит экономить на качестве.

Считайте первый цикл в создании эволюционного прототипа пробным выпуском, реализующим начальную порцию требований. Уроки, извлеченные из реакции пользователей на тестирование и первоначальное использование продукта, учитываются при модификации в следующем цикле. Окончательный продукт — это кульминация серии эволюционных прототипов. Такие прототипы позволяют пользователям быстро получить действующие образцы. Эволюционные прототипы хорошо подходят для приложений, которые, как вы знаете, со временем будут расширяться, но они также представляют ценность для пользователей, у которых нет реализации всей запланированной функциональности. Проекты гибкой разработки часто планируют так, чтобы разработку можно было остановить в конце итерации и все равно получить полезный для клиентов продукт, несмотря на свою незавершенность.

Эволюционные прототипы хорошо подойдут для проектов разработки интернет-приложений. В одном из таких проектов, которым я руководил, моя команда создала серию из четырех прототипов на основе требований, полученных в результате анализа вариантов использования. Каждый прототип оценивали несколько пользователей, и мы вносили изменения на основе их ответов на наши вопросы. Исправления, которые мы внесли после оценки четвертого прототипа, стали окончательным вариантом нашего интернет-сайта.

На рис. 15-1 показано несколько способов комбинирования различных видов прототипов. Например, информацию, полученную в результате создания серии одноразовых прототипов, можно использовать для уточнения требований, которые затем шаг за шагом реализовать через последовательность эволюционных прототипов. Другой способ, показанный на рис. 15-1, — применение одноразовой модели для прояснения требований перед разработкой окончательной версии дизайна пользовательского интерфейса, в то время как одновременно с помощью экспериментального образца проверяется архитектура системы и базовые алгоритмы. Что вам совершенно точно *не удастся,* так это обеспечить в намеренно низкокачественном одноразовом прототипе устойчивость, необходимую для производственной системы. В дополнение, рабочие прототипы, которые демонстрируют возможности продукта нескольким пользователям, привлекаемым к разработке системы, скорее всего нельзя масштабировать до тысяч пользователей без серьезных архитектурных изменений. В табл. 15-1 суммированы некоторые стандартные варианты применения моделей, экспериментальных образцов и одноразовых и эволюционных прототипов.

Табл. 15-1. Типичные применения прототипов ПО

	Одноразовый	**Эволюционный**
Модель	Прояснение и уточнение пользовательских и функциональных требований Выявление пропущенной функциональности Исследование возможных вариантов пользовательского интерфейса	Реализация базовых пользовательских требований Реализация дополнительных пользовательских требований по приоритетам Реализация и доработка веб-сайтов Адаптация системы к быстро меняющимся требованиям бизнеса
Экспериментальный образец	Демонстрация технической осуществимости Оценка производительности Получение информации для уточнения оценок	Реализация и наращивание ключевой многоуровневой функциональности и уровней коммуникации Реализация и оптимизация основных алгоритмов Тестирование и настройка производительности

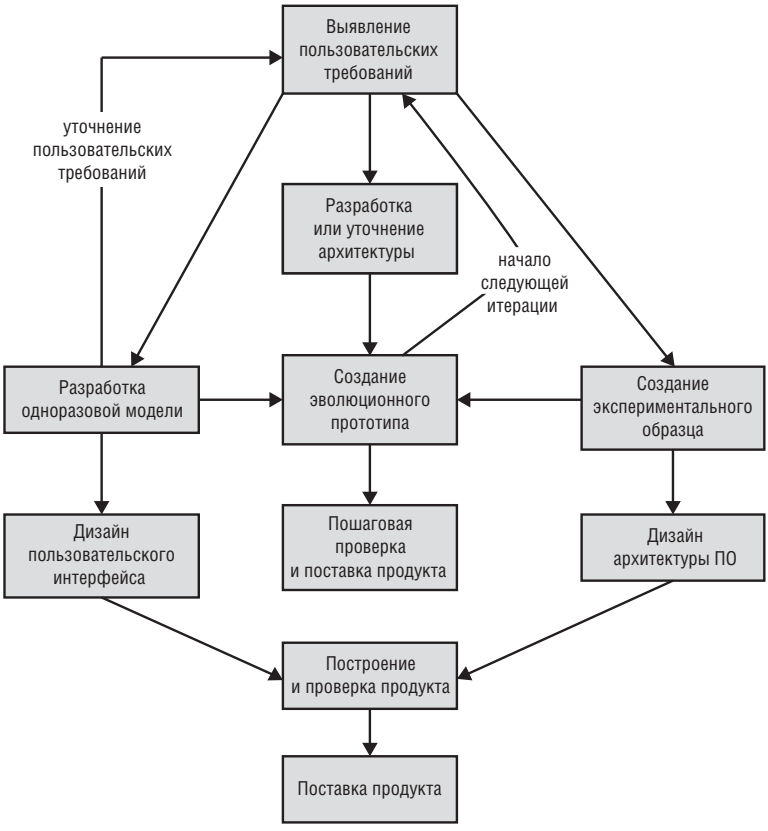


Рис. 15-1. Несколько возможных способов использования прототипов в процессе разработки ПО

Бумажные и электронные прототипы

Не всегда для разрешения неопределенностей в требованиях нужен прототип в виде исполняемого кода. *Бумажный прототип* (paper prototype) (иногда его называют низкокачественным прототипом) — это дешевый, быстрый и низкотехнологичный способ выяснить, как может выглядеть некий фрагмент системы (Rettig, 1994). Бумажные прототипы помогают установить, действительно ли пользователи и разработчики одинаково понимают требования. Они позволяют вам попробовать, практически не рискуя, сделать пробный шаг в пространство возможных решений продукта до разработки производственного кода продукта. Похожий метод, называемый методом *раскадровки* (storyboard) (Leffingwell и Widrig, 2000), показывает предлагаемый интерфейс пользователя, без привлечения пользователей к работе с ним. Используйте низкокачественные прототипы для изучения функциональности и потоков, а высококачественные — для определения точного внешнего вида и поведения.

Для создания бумажных прототипов требуются инструменты не сложнее, чем бумага, карточки, наклейки и доска. Дизайнер делает наброски экранов, как он их представляет, не заботясь о том, где точно будут располагаться элементы управления и как они будут выглядеть. Пользователи с готовностью делятся своим мнением о нарисованном на бумаге, но в некоторых случаях они не склонны критиковать полюбившийся им компьютерный прототип, в который разработчик, по всей видимости, вложил много труда. Разработчики также иногда противятся внесению существенных изменений в тщательно выполненный электронный прототип.

При оценке низкокачественного прототипа кто-то играет роль компьютера, а пользователь проходит оценочный сценарий. Он инициирует действия, произнося вслух, что он собирается сделать в конкретном экране: «Я хочу выбрать команду Предварительный просмотр из меню Файл». При этом тот, кто играет роль компьютера показывает страницу или карточку, представляющую экран, который должен появиться после выполнения действия. Это позволяет оценить, соответствует ли ответ ожидаемому и содержит ли экран необходимые элементы. Если нет, вы просто берете чистый лист бумаги или карточку и рисуете все заново.

Отправляемся к доброму волшебнику!

Команда разработчиков, создававшая большие копировальные устройства, однажды пожаловалась мне, что у их последнего устройства обнаружились проблемы с удобством использования. Самая обычная задача по копированию выполнялась в пять действий, что пользователям казалось ужасно неудобным. «Надо было нам сделать прототип прежде, чем создавать продукт», — тоскливо сказал один из разработчиков.

Но как же создавать прототип такого сложного продукта, как копировальный аппарат? Для начала купите большой холодильник. После этого,

напишите на коробке из-под холодильника *КОПИРОВАЛЬНЫЙ АППАРАТ*. Потом посадите кого-нибудь внутрь коробки и попросите пользователя подойти снаружи и изображать различные операции с аппаратом. Человек внутри коробки будет реагировать так, как, по его мнению, реагировало бы устройство, а представитель пользователей должен отмечать, та ли это реакция, которую он себе представляет. С помощью простого, веселого прототипа, подобного этому, — иногда его называют «прототипом волшебника из страны Оз» — зачастую удается на ранних стадиях разработки получить отзывы пользователей, руководствуясь которой разработчики и разрабатывают решения. Плюс вы получаете большой холодильник.

Какими бы совершенными ни были ваши инструменты создания прототипов, наброски экранов на бумаге или на доске делать все равно быстрее. Бумажные прототипы ускоряют каждый цикл, а это — ключевой фактор успеха при разработке требований. Это отличный метод уточнения требований перед проектированием детализированных интерфейсов пользователей, создания эволюционных прототипов или традиционного дизайна и проектирования. Он также помогает команде лучше управлять ожиданиями клиентов.

Если вы решите создать электронный одноразовый прототип, вам на выбор предлагается много соответствующих инструментов: от средств рисования, таких как Microsoft Visio и Microsoft PowerPoint, до серийных инструментов создания прототипов и графических конструкторов графического пользовательского интерфейса. Есть также специализированные инструменты для создания каркасов веб-сайтов. Такие инструменты позволят вам с легкостью реализовать и обновлять компоненты интерфейса пользователя, вне зависимости от неэффективности стоящего за ним кода. Конечно же, если вы создаете эволюционный прототип, то должны использовать высококачественные средства разработки с самого начала. Так как инструменты и их разработчики меняются очень быстро, мы не станем рекомендовать какое-то конкретное средство.

На рынке существует много инструментов, позволяющих моделировать приложение до его построения. Моделирование приложений позволяет быстро собирать макеты страниц, элементы управления пользовательского интерфейса, путей навигации и функциональности в что-то, что очень похоже на продукт, который предполагается создать. Возможность пошагового моделирования предоставляет ценный механизм взаимодействия с представителями пользователей над уточнением требований и пересмотру решения.

Какой бы тип прототипов не использовался — бумажные или электронные прототипы или модели — бизнес-аналитик должен четко следить за тем, чтобы не оказаться раньше времени вовлеченным в планирование детальных подробностей пользовательского интерфейса. Те, кто оценивает прототип, часто дают советы, например: «Можно сделать этот текст более темным красным?» или «Мне не нравится этот шрифт». Подобные замечания бесполезны и лишь отвлекают, если только задача прототипа не состоит в разработке

подробного дизайна экранной формы или веб-страницы. Цвет, шрифт и размещение полей несущественны, если приложение не обеспечивает надлежащую поддержку бизнес-задач пользователя. Пока вы не достигли высокого уровня понимания необходимой функциональности, направьте работу с прототипами на уточнение требований, а не видимого дизайна.

Работа с прототипами

На рис. 15-2 показана одна возможная последовательность шагов разработки от вариантов использования до детализированного дизайна интерфейса пользователя через построение одноразового прототипа. Каждое описание варианта использования включает последовательность действий субъекта и реакцию системы, которые вы можете смоделировать посредством карты диалоговых окон, чтобы отразить особенности архитектуры интерфейса пользователя. Одноразовый прототип или каркас представляет элементы диалога в виде экранов, меню и диалоговых окон. Когда его оценивают пользователи, возможны изменения в описании вариантов использования (если, например, обнаруживается альтернативный путь) или изменения в карте диалоговых окон. Каждый раз, когда требования уточняются и создаются наброски экранов, элементы интерфейса пользователя оптимизируются для удобства и простоты использования. Эти действия не обязательно выполнять точно в указанной последовательности. Постепенное уточнение варианта использования, карты диалоговых окон и каркаса — наилучший способ быстро получить доступный и согласованный дизайн пользовательского интерфейса.

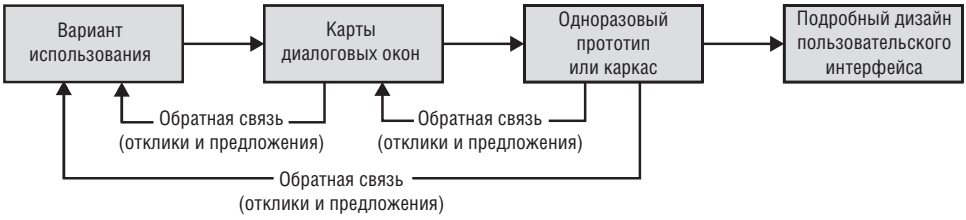


Рис. 15-2. Последовательность действий от вариантов использования к дизайну интерфейса пользователя через одноразовый прототип

Такой метод постепенного уточнения обходится дешевле, чем если вы перескочите прямо от описаний вариантов использования к законченному интерфейсу пользователя, а потом обнаружите серьезные проблемы в требованиях, исправление которых потребует глобальной переделки продукта. Надо лишь в этой последовательности выполнить столько шагов, сколько необходимо для снижения до допустимого уровня риска неправильной разработки дизайна пользовательского интерфейса. Если в вашей команде уверены, что понимают требования и интерфейс, тогда требования можно считать в целом готовыми и особого смысла в создании прототипов нет. Также прототипы можно сориентировать на пользовательские требования, в которых высока

вероятность или цена ошибки. В одном проекте переделывался дизайн веб-сайта крупной корпорации, который посещают миллионы пользователей. В команде создали прототипы основных элементов веб-сайта, включая онлайновый каталог, корзину для покупок и процесс оформления заказа, чтобы все сделать правильно с первого раза. Они потратили не очень много времени на изучение исключительных ситуаций и менее популярных сценариев поведения.

Чтобы нагляднее понять весь процесс, посмотрим на реальный пример небольшого веб-сайта для продвижения книги мемуаров о жизненных уроках под названием «Pearls from Sand» («Жемчужины из песка»). Автор книги (его зовут Карл) подумал о некоторых вещах, которые посетители сайта должны иметь возможность делать, и сформулировал соответствующие варианты использования. Есть еще варианты использования для других классов пользователей (табл. 15-2).

Табл. 15-2. Некоторые варианты использования для сайта PearlsFromSand.com

Класс пользователей	Вариант использования
Посетитель	Получить информацию о книге
	Получить информацию об авторе
	Читать избранные главы
	Читать блог
	Связаться с автором
Клиент	Заказать продукт
	Загрузить электронную версию продукта
	Обратиться за помощью в решении проблемы
Администратор	Управлять списком продуктов
	Осуществить возврат средств клиенту
	Управлять списком сообщений электронной почты

Следующий этап — обдумывание, какие страницы должны быть на веб-сайте, и представить пути навигации между ними. В окончательной версии не обязательно все эти страницы будут отдельными — одни будут объединены, а другие станут всплывающими страницами или другими частями другой страницы. На рис. 15-3 показана часть карты диалоговых окон, иллюстрирующая концепцию архитектуры страниц. Каждый прямоугольник представляет страницу, которая будет участвовать в операциях, описанных в вариантах использования. Стрелки представляют ссылки, обеспечивающие навигацию между страницами. В процессе рисования карты диалоговых окон могут обнаруживаться новые действия, которые пользователь может захотеть выполнить. При работе над вариантом использования могут обнаружиться возможности упростить и улучшить работу пользователя.

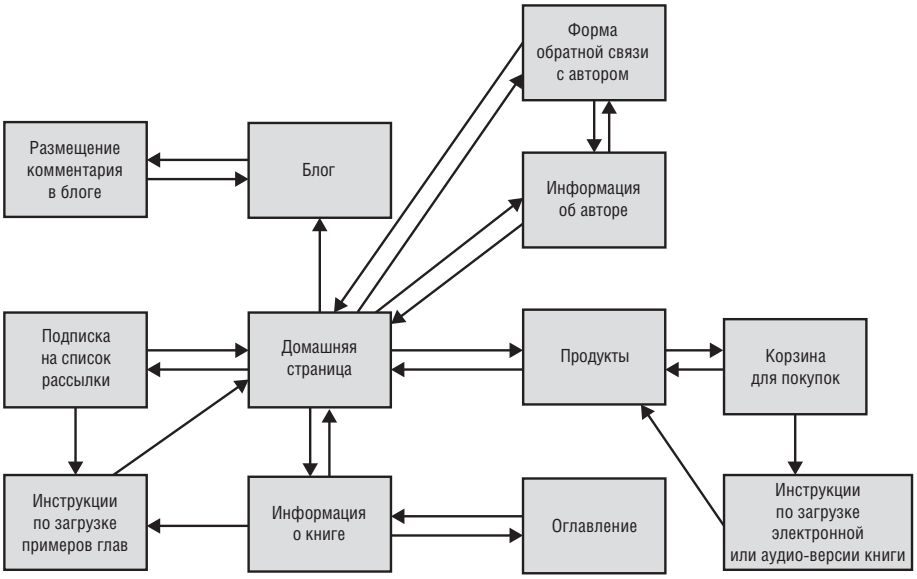


Рис. 15-3. Фрагмент карты диалоговых окон сайта PearlsFromSand.com

Следующий этап — построение одноразового прототипа или каркаса избранных страниц для выработки концепции визуального дизайна. Каждую страницу можно набросать от руки на бумаге (см. пример на рис. 10-1 в главе 10) или создать модель с помощью специального средства визуального дизайна или создания прототипов.

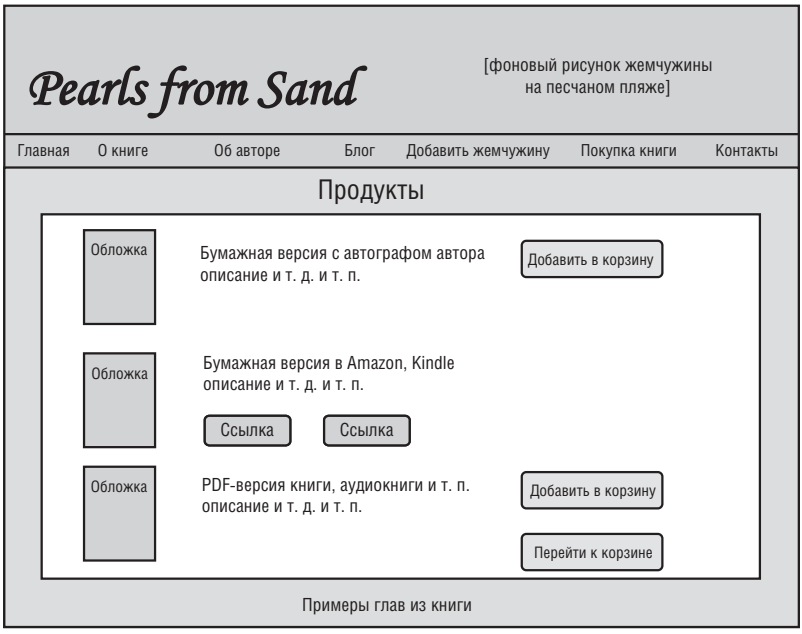


Рис. 15-4. Пример каркаса одной страницы сайта PearlsFromSand.com

Каркас на рис. 15-4 был создан за несколько минут с помощью PowerPoint. Такая простая диаграмма — средство, которое можно использоваться для работы с представителями пользователей, чтобы в целом понять, какой макет страницы и косметические свойства сделают страницу простой в понимании и использовании.

Наконец, четвертый этап, показанный на рис. 15-2, — создание подробного дизайна пользовательского интерфейса. На рис. 15-5 показана одна окончательная страница веб-сайта PearlsFromSand.com — конечная точка описанных выше анализа требований и работы с прототипами. Такой поэтапный подход к дизайну пользовательского интерфейса позволяет получить более качественные результаты, чем немедленный переход к подробному дизайну страницы без четкого понимания того, что могут хотеть представители разных классов пользователей сделать при посещении веб-сайта.

Рис. 15-5. Финальная реализация страницы сайта PearlsFromSand.com

Оценка прототипа

Оценка прототипа связана с тестированием удобства использования (Rubin и Chisnell, 2008). Вы узнаете больше, наблюдая за работой пользователей с прототипом, чем просто спрашивая их мнение. Смотрите, к каким клавишам инстинктивно тянутся пальцы пользователя или перемещается указатель мыши. Отмечайте места, где прототип конфликтует с другими приложениями, с которыми часто работают пользователи. Человек, оценивающий про-

тотип может пытаться использовать неправильные сочетания клавиш или водить указателем мыши в поиске нужной команды меню. Отмечайте нахмуренные брови, свидетельствующие, что пользователь озадачен и не знает, что делать дальше, как добраться до нужной точки или открыть другую часть приложения, чтобы просмотреть еще что-нибудь. Посмотрите, нет ли в прототипе тупиков, как иногда бывает, когда пользователь щелкает кнопку отправки формы на веб-сайте.

Удостоверьтесь, что пользователи оценивали прототип с соответствующих точек зрения. Привлекайте пользователей многих классов, как опытных пользователей, так и новичков. Показывая прототип тем, кто его будет оценивать, подчеркните, что в нем воплощена лишь часть нужных функций, остальные будут реализованы в конечной системе.

> **Внимание!** Как и при любом другом тестировании удобства использования, следите за тем, чтобы не забыть о членах важных классов пользователей и не исключить их из оценки прототипа. Новичку прототип может понравиться простотой работы с ним, но более опытного пользователя может раздражать тем, как прототип тормозит его работу. Постарайтесь не забыть об обеих группах пользователей.

Для улучшения оценки горизонтальных прототипов создавайте сценарии, проводящие пользователей через последовательность действий, и задавайте конкретные вопросы, чтобы выявить требуемую информацию. Этот метод — дополнение к общему приглашению: «Расскажите, что вы думаете об этом прототипе». Сценарии оценки составляйте на основании вариантов использования, пользовательских историй или функций, которые должен представлять прототип. Сценарий попросит пользователей выполнить определенные задачи и проведет их через наиболее неясные области прототипа. В конце выполнения каждой задачи и, возможно, в промежуточных пунктах, сценарий задает связанные с задачами вопросы. Ну и, конечно же, вы вправе задать вопросы сами, например такие:

- Реализует ли прототип все необходимые функции так, как вы этого ожидали?
- Не упущена ли в прототипе какая-либо функциональность?
- Заметили ли вы какие-либо условия ошибки, не учтенные в прототипе?
- Нет ли в прототипе каких-либо ненужных функций?
- Насколько логичной и полной вам кажется навигация?
- Не показалось ли вам, что можно упростить каких-либо задачи, в которых слишком много интерактивных шагов?
- Оказывались ли вы в ситуации, когда не знали, что делать дальше?

Просите ваших помощников вслух выражать их мнение во время работы с прототипом, чтобы вы могли понимать ход их мыслей и отметить требования, которые прототип обрабатывает плохо. Создайте дружелюбную атмосферу, в которой пользователи будут свободно высказывать свои идеи и замечания. Избегайте показывать им «правильный» путь реализации тех или иных функций в прототипе.

Записывайте все, что узнаете в процессе оценки прототипа. Модель позволит вам уточнить спецификацию требований к ПО. Если на основе оценки прототипа были приняты какие-либо решения о дизайне пользовательского интерфейса или выбраны конкретные методы взаимодействия, запишите эти выводы и то, как вы к ним пришли. Иначе вам придется снова и снова возвращаться к одному и тому же — пустая трата времени. Для экспериментального образца задокументируйте процесс и результаты оценки, в том числе решения, которые вы приняли об исследованных технических процессах. Разрешите все несоответствия между спецификацией требований к ПО и прототипом.

Риски создания прототипов

На создание даже простого прототипа нужны время и деньги. Хотя создание прототипов и снижает риск провала проекта разработки ПО, оно влечет за собой собственные риски, некоторые из которых объясняются в этом разделе.

Подталкивание к выпуску прототипа

Наибольший риск заключается в том, что кто-либо из заинтересованных в проекте лиц, увидев действующий одноразовый прототип, решит, что окончательный продукт почти готов. «Ого, да вы, похоже, уже все сделали! — с энтузиазмом может сказать тот, кого вы попросили оценить прототип. — Выглядит отлично. Может, вы быстро доделаете его и отдадите мне?»

Если коротко, то: НЕТ! Одноразовый прототип ни в коем случае не предназначен для работы, как бы он ни был похож на готовый продукт. Это всего лишь модель, образец, эксперимент. Если нет жестких коммерческих обстоятельств, заставляющих немедленно застолбить место на рынке (в этом случае руководство понимает и принимает связанный с этим риск получить страшно недовольных пользователей), сопротивляйтесь всем, кто настаивает на поставке одноразового прототипа в качестве готового продукта. Такой шаг в действительности лишь отдалит сроки завершения продукта, потому что и структура, и код одноразового прототипа намеренно создавались без учета качества или надежности. Управление ожиданиями — ключ к успешному созданию прототипов. Каждый, кто видит прототип, должен понимать его цель и ограничения. Четко понимайте, почему вы создаете определенные типы прототипов, решайте, какая их в конечном итоге ждет судьба и точно и внятно доводите это до заинтересованных лиц, работающих с этими прототипами.

Не позволяйте страхам, связанным с преждевременным выпуском продукта, отвратить вас от создания прототипа. Объясните всем, что вы не выпустите его как окончательный продукт. Один из способов контролировать этот риск — использовать бумажные, а не электронные прототипы. Ни одному из тех, кто оценивает бумажный прототип, и в голову не придет сказать, что продукт уже почти готов! Еще один способ — выбрать инструменты создания

прототипов, отличающиеся от тех, что применяются для разработки окончательного продукта. Никто не воспримет модель навигации в PowerPoint или простой каркас как что-то реальное. Это поможет противостоять тем, кто просит «быстро закончить» и выпустить прототип. Оставив внешний вид прототипа неотделанным и незаконченным, вы также уменьшите этот риск. Некоторые из многих инструментов для создания каркасов позволяют быстро разрабатывать высококачественный пользовательский интерфейс. Это повышает вероятность того, что люди начнут относиться к прототипу как к почти готовому продукту и это усилит давление, чтобы заставить вас преобразовать одноразовый прототип в эволюционный.

Один разработчик набросал работающий прототип пользовательского интерфейса в кричащем розовом цвете. Вот что он рассказал: «Когда мы показали клиентам первые несколько итераций с этой цветовой схемой, НИ У КОГО не сложилось впечатление, что это может быть практически готовый продукт. На самом деле я оставил этот ужас еще на одну итерацию — просто чтобы не попасть в одну из ловушек демонстрации красивых прототипов клиенту».

Отвлечение на детали

Еще один риск создания прототипов возникает, когда пользователи начинают зацикливаться на деталях того, как интерфейс пользователя будет выглядеть и действовать. Работая с прототипами, внешне похожими на окончательный продукт, пользователи легко забывают, что на стадии уточнения требований они должны в основном думать о концептуальных вопросах. Ограничьте прототип только теми экранами, функциями и возможностями навигации, которые помогут устранить неопределенности в требованиях.

Не выплеснуть с водой младенца

Я как-то консультировал компанию, где большой начальник запретил прототипы. Он видел проекты, в которых клиенты заставляли разработчиков раньше времени выпускать одноразовые прототипы как конечный продукт — с ожидаемыми результатами. Прототипы не обрабатывали ошибки пользователей или неправильные входные данные, не предоставляли все нужные пользователю функции и были сложны для поддержки и сопровождения. На основе этого неприятного опыта начальник решил, что прототипы — прямой путь к проблемам.

Как вы видели в этой главе, предоставление клиентам прототипа, который планировалось выбросить, в качестве продукта очевидным образом вызывает проблемы. Тем не менее, создание прототипов предоставляет ряд мощных приемов, которые могут внести существенный вклад в построение правильного продукта. Вместо того, чтобы отказываться от прототипов как от опасного метода, которого нужно избегать, важно сделать так, чтобы все участники понимали, какие бывают прототипы, зачем они создаются и как можно использовать результаты.

Нереалистичные ожидания производительности

Третий риск состоит в том, что пользователи начнут делать выводы о производительности конечного продукта по производительности прототипа. Но оценка модели не проводится в рабочей среде продукта. Кроме того в прототипе могут применяться менее эффективные средства, например интерпретируемые сценарии, а не скомпилированный код. В экспериментальном образце не всегда применяются отлаженные алгоритмы или уровни защиты, что скажется на конечной производительности. Если пользователи увидят, что прототип мгновенно реагирует на моделируемый запрос к базе данных, используя жестко закодированные результаты запроса, они могут ждать такой же поразительной производительности от продукта, работающего с огромной распределенной базой данных. Подумайте о реализации в прототипе временных задержек, чтобы модель ожидаемого поведения окончательного продукта выглядела более реалистичной (а прототип — менее готовым для реализации). Можно разместить на экране сообщение, которое четко говорит, что поведение готовой системы может отличаться.

При гибкой разработке или других ситуациях использования эволюционных прототипов нужно гарантировать создание устойчивой и расширяемой архитектуры и с самого начала разрабатывать код высокого качества. Вы создаете производственное ПО, но небольшими порциями. Можно скорректировать дизайн посредством переделки в последующих итерациях, но не нужно откладывать и заменять необходимость обдумывания дизайна сегодня на переделку в будущем.

Слишком много усилий на создание прототипов

Наконец, опасайтесь действий по созданию прототипов, требующих таких усилий, что команда разработчиков выбьется из графика и будет вынуждена выпустить прототип в качестве готового продукта или торопливо и бессистемно доделывать продукт. Это возможно, когда создается прототип всего решения, а не только самых неочевидных, высокорискованных или сложных его частей. Относитесь к прототипу, как к эксперименту. Вы проверяете гипотезу, что требования определены в достаточном объеме, и что ключевые аспекты взаимодействия человека и компьютера, а также все архитектурные вопросы решены, так что можно переходить к проектированию и конструированию. Возможностей у прототипа должно быть ровно сколько, сколько необходимо для получения ответов на вопросы и уточнения требований.

Факторы успеха использования прототипов

Использование прототипов ПО предлагает мощный набор методов, которые могут позволить сократить сроки разработки, удовлетворить клиентов и создать продукты высокого качества. Чтобы сделать прототипы эффективной

частью процесса составления требований, прислушайтесь к следующим рекомендациям:

- включите задачи создания прототипов в план своего проекта. Составьте график распределения затрат времени и средств на разработку, оценку и модификацию прототипов;
- сформулируйте цель каждого прототипа до начала его создания и объясните, что произойдет с ним в будущем: прототип будет отброшен (или перемещен в архив), а полученная с его помощью информация будет сохранена, или он вырастет в конечное решение. Убедитесь, что те, кто строят прототипы и кто оценивает их, понимают ваши намерения;
- планируйте создание нескольких прототипов. В большинстве случаев вам не удастся создать то, что нужно, с первой попытки (в чем, собственно, и состоит весь смысл создания прототипов!);
- одноразовые прототипы создавайте так быстро и дешево, как возможно. Вкладывайте минимум усилий в подготовку прототипов, которые помогут ответить на вопросы или разрешить неясности в требованиях. Не доводите до совершенства одноразовые прототипы;
- не встраивайте в одноразовый прототип подробную проверку входных данных, методы безопасного программирования, обработку ошибок или документацию в коде. Это ненужные затраты на то, что в конечном итоге будет отброшено;
- не создавайте прототипы элементов, которые уже понимаете, кроме случаев исследования альтернативных вариантов дизайна;
- используйте правдоподобные данные в экранных формах и сообщениях прототипа. Пользователи будут отвлекаться на нереалистичные данные и не смогут воспринять прототип как модель, описывающую внешний вид и поведение реальной системы;
- не ожидайте, что прототип полностью заменит спецификацию требований к ПО. Большая часть скрытой функциональности лишь подразумевается в прототипе и должна быть задокументирована в спецификации требований к ПО, чтобы ее удалось реализовать полно, точно и прозрачно. Изображения экранов не дают деталей определения полей данных, критериев проверки, взаимосвязей между полями (таких, как элементы управления пользовательского интерфейса, появляющихся только если пользователь совершает определенные действия с другими элементами управления), обработки исключений, бизнес-правил и другой жизненно важной информации.

Продуманное применение и грамотная реализация прототипов служит ценным инструментом, помогающим выявлять и уточнять требования в этом сложном деле преобразования потребностей в решения.

Что дальше?

- Выделите часть вашего проекта, в котором наблюдается путаница в требованиях или есть функциональность, характеризующаяся высоким риском. Набросайте часть возможного интерфейса пользователя, который представляет ваше понимание требований и того, как они должны быть воплощены, — бумажный прототип. Попросите нескольких пользователей выполнить на вашем прототипе сценарий использования. Выявите места, где первоначальные требования были неполными или неверными. Измените соответствующим образом прототип и проведите повторный анализ, чтобы удостовериться, что недостатки устранены.
- Кратко перескажите содержание этой главы тем, кто будет оценивать ваш прототип, чтобы помочь им понять логику работы с прототипами и настроиться на реалистичные ожидания результатов.
- Если вы создаете аппаратное устройство, подумайте, как физически смоделировать его, чтобы пользователи смогли взаимодействовать с ним и уточнить свои требования.

Глава 16

Сначала о главном: определение приоритетов требований

После того как большинство требований к Chemical Tracking System было определено, менеджер проекта Дэйв и бизнес-аналитик Лори встретились с двумя сторонниками продуктов. Тим представлял химиков, а Роксана выступала от лица работников склада химикатов.

Дейв сказал: «Теперь, когда мы в целом понимаем, какие основные функции вам нужны, нам нужно подумать о выборе части выявленных пользовательских историй для реализации в первых итерациях. Важно, чтобы мы достигли согласия о том, с чего начать, чтобы вы могли как можно скорее пользоваться частью функциональности системы. Давайте проведем начальное определение приоритетов этих пользовательских историй, чтобы мы знали, что для вас самое важное. Тогда мы сможем понять в точности, что вы ожидаете от каждой из этих начальных функций».

Тим был озадачен. «Зачем вам назначать приоритеты требований? Все требования важны, иначе бы мы не дали их вам».

Лори, бизнес-аналитик, пояснил: «Мы знаем, но мы должны обеспечить, чтобы самые необходимые срочные требования вошли в первые итерации. Мы просим вас помочь нам отделить требования, которые необходимо выполнить первыми, от тех, которые могут подождать до более поздних итераций. Не могли бы вы указать функциональность, которая при реализации в первую очередь принесла бы самую большую пользу химикам и другим классам пользователей?»

«Я знаю, что отчеты, которые отдел охраны труда и техники безопасности должен составлять для регулирующих органов, должны быть готовы в ближайшее время или у компании будут неприятности, — заметала Роксана. — Мы можем использовать нашу нынешнюю систему учета еще несколько месяцев, если придется».

Тим запротестовал: «Я обещал химикам функцию поиска по каталогу в качестве примера того, как эта система будет экономить им время. Не могли бы мы начать с него? Он может быть не идеальным, но нам нужен доступ к каталогам как можно быстрее».

Тим и Роксана поняли, что ожидать чудес неразумно и если продукт не может содержать все требования в версии 1.0, то для всех будет лучше, если удастся согласовать первоочередные требования. Они продолжили разбор пользовательских историй, разбив их на первоочередные и те, что могут подождать.

В немногих проектах разработки ПО удается реализовать все затребованные заинтересованными лицами функции к дате начального выпуска. Для каждого проекта, на реализацию которого выделены ограниченные ресурсы, необходимо определить относительные приоритеты затребованных функций продукта. Расстановка приоритетов, которую еще называют *разбором требований* (requirements triage) (Davis, 2005), помогает выявлять взаимоисключающие цели, разрешать конфликты, планировать промежуточные и инкрементальные выпуски, управлять расползанием границ проекта и принимать необходимые компромиссы. В этой главе рассказано о важности определения приоритетов требований, описаны несколько приемов приоритизации и предлагается таблица для анализа расстановки приоритетов на основе ценности, затрат и рисков.

Зачем определять приоритеты требований

Когда ожидания клиентов высоки, а сроки поджимают, вам нужно, чтобы в продукте были реализованы самые критические или ценные функции как можно раньше. Приоритеты — это способ разрешения борьбы между конкурирующими требованиями за ограниченные ресурсы. Определение относительного приоритета каждой функции позволяет вам так планировать разработку, чтобы обеспечивать наибольшую ценность при наименьших затратах. Так как приоритеты определяются относительно друг друга, приоритизацию можно начинать сразу же после выявления второго требования.

Иногда клиентам не нравится определять приоритеты, потому что полагают, что никогда не получат реализацию требований с низким приоритетом. Ну, если вы не получаете все, что хотели — как обычно и происходит, — то наверняка должны позаботиться о том, чтобы *точно* получить самые важные функции для достижения своих бизнес-целей. Иногда разработчики ленятся определять приоритеты требований, потому что им кажется, что они легко реализуют их все. Но реальность такова, что это не так, по крайней мере не все требования сразу. Расстановка приоритетов помогает проекту обеспечить максимальную пользу для бизнеса как можно быстрее и в рамках ограничений проекта.

Определение приоритетов — критически важная стратегия в проектах гибкой разработки и в других проектах, где продукты разрабатываются как последовательность фиксированных ограниченных во времени итераций. Проектные команды могут пополнять резерв своего продукта пользовательскими историями, функциями, бизнес-процессами и историями дефектов (то есть тех, что ожидают исправления). Клиенты расставляют приоритеты

этих историй в резерве проекта и отбирают те, что должны реализовываться в каждой итерации разработки. Разработчики оценивают усилия, необходимые для реализации каждой истории и прикидывают, сколько историй можно разместить в той или иной итерации, основываясь на эмпирически определенной производительности. При появлении новых историй клиенты могут переоценивать приоритеты в резерве проекта, таким образом динамически корректируя объем будущих итераций. Это надо делать во всех проектах, чтобы гарантировать, что команда всегда работает над теми функциями, которые позволят максимально быстро предоставить пользователям полезное ПО.

В каждом проекте менеджер должен сбалансировать желаемый объем проекта и ограничения, определяемые срокам, бюджетом, людскими ресурсами и целями по качеству (Wiegers, 1996). Один из способов достижения этого — убрать (или отложить до более поздней версии) требования с низким приоритетом, когда принимаются новые, более важные требования или изменяются другие условия проекта. То есть определение приоритетов это динамический и постоянный процесс. Если клиенты не классифицируют свои требования по важности и срочности, менеджерам проектов приходится делать это своими силами. Не удивительно, что клиентам не всегда нравится результат; поэтому они должны указывать, какие требования необходимы с самого начала, а какие могут подождать. Определяйте приоритеты на ранних стадиях проекта, когда больше шансов на успешный результат, и периодически возвращайтесь к ним.

Достаточно сложно заставить даже одного клиента решить, какие из его требований приоритетнее. Согласовать мнения нескольких клиентов с различными ожиданиями еще труднее. Люди по природе склонны отстаивать свои интересы и не жаждут жертвовать собственными нуждами ради чужой выгоды. Тем не менее участие в определении приоритетов требований — одна из обязанностей клиента в отношениях клиент-разработчик, как уже говорилось в главе 2. Обсуждение приоритетов помогает не только определить последовательность воплощения требований, но и прояснить ожидания клиентов.

Некоторые практические соображения определения приоритетов

Даже проект средних масштабов может иметь десятки пользовательских требований и сотни функциональных требований — слишком много, чтобы классифицировать их аналитически и последовательно. Чтобы не потерять контроль над ситуацией, выберите для определения приоритетов определенный уровень абстракции — функции, варианты использования, пользовательские истории или функциональные требования. В рамках одного варианта использования некоторые альтернативные направления могут иметь больший приоритет, чем остальные. Вы имеете возможность сначала

определить приоритеты на уровне функций, а затем назначить приоритеты функциональным требованиям отдельно для каждой функции. Это поможет вам отделить основную функциональность от усовершенствований, которые можно реализовать позже или вообще убрать. Как описано в главе 5, определение приоритетов функций непосредственно влияет на планирование границ и выпуска проекта. Не стоит забывать и о низкоприоритетных требованиях, хотя пока нет причин анализировать их более подробно. Их приоритет может впоследствии измениться, и если разработчики знают о них сейчас, им проще планировать будущую модификацию.

В определении приоритетов должны участвовать разные заинтересованные лица, представляющие клиентов, кураторов, руководство проекта, разработчиков и других. Обязательно должен быть один ответственный за принятие решения на тот случай, если заинтересованные лица не могут достичь согласия. Хорошей отправной точкой для участников приоритизации является достижение согласия насчет критериев, которые будут применяться для определения, когда одно требование обладает более высоким приоритетом, чем другое. В процессе определения приоритетов могут применяться такие характеристики, как ценность для клиента, бизнес-ценность, технические или бизнес-риски, затраты, сложность реализации, время выпуска на рынок, соответствие нормативам или политикам, конкурентное преимущество и договорные обязательства (Gottesdiener, 2005). Алан Дейвис (Alan Davis, 2005) указывает, что для успешного определения приоритетов требуется понимание шести характеристик:

- потребностей клиентов;
- относительной важности требований для клиентов;
- последовательности, в которой должны предоставляться функции;
- требований, служащих основой для других требований, и других связей между требованиями;
- потребности реализовывать какие-то группы требований целиком;
- затраты на удовлетворение каждого требования.

Клиентам больше всего нужны функции, наиболее ценные для бизнеса или удобства работы. Однако, когда разработчики обрисуют затраты, трудоемкость, технический риск или компромиссы, связанные с каждым требованием, клиенты могут передумать и прийти к выводу, что это требование не так важно, как они считали изначально. Разработчики же иногда решают на ранних стадиях реализовать некоторые функции с низким приоритетом, потому что они влияют на архитектуру системы и создают фундамент для эффективной реализации будущей функциональности без серьезной перестройки. У части функциональности должен быть высокий приоритет, потому что она нужна, чтобы приложение удовлетворяло требованиям регулирующих органов. Как всегда при разработке требований, перекрывающиеся бизнес-цели, которые собственно и вызвали к жизни проект, должны определять решения о приоритетах.

Некоторые требования должны реализовываться вместе или в определенной последовательности. Нецелесообразно реализовывать функцию повтора правки в первом выпуске, но не реализовывать парную функцию отмены правки на протяжении нескольких месяцев. Аналогично, представьте, что вы реализовали нормальное направление определенного варианта использования в первом выпуске, а менее приоритетные альтернативные направления отложили до более поздней даты. Это нормально, но вы должны также реализовать соответствующие обработчики исключений одновременно с реализацией всех успешных направлений. В противном случае может оказаться, что вы, скажем, пишете код обработки платежей с использованием кредитных карт без проверки действительности этих карт, отклонения карт, объявленных украденными, или обработки других исключений.

Игры, в которые люди играют с приоритетами

Естественная реакция пользователей на просьбу определить приоритеты обычно такая: «Мне нужны все эти функции. Уж как-нибудь реализуйте их». Они чувствуют, что каждому требованию нужно назначить высокий приоритет, и не могут не осознавать, что определение приоритетов позволит обеспечить успех проекта. Начните с объяснения, что все нельзя сделать сразу, поэтому вы хотите убедиться, что в первую очередь вы займетесь самыми нужными вещами. Трудно убедить клиентов обсуждать приоритеты, если они знают, что функций с низким приоритетом не получат никогда. Один разработчик как-то сказал мне, что в их компании не принято назначать требованию низкий приоритет. Категории приоритетов требований назывались у них «высокие», «очень высокие» и «исключительно высокие». Другой разработчик, выполняющий роль бизнес-аналитика, заявлял, что назначение приоритетов совсем не нужно: если он записал что-либо в спецификацию требований, то собирается это реализовать. Но это не отвечает на вопрос, *когда* будет реализована каждая функция.

Недавно я был в одной компании, сотрудникам которой очень трудно давалась своевременная реализация проектов. Хотя руководство и заявляло, что будет несколько выпусков приложений, чтобы менее приоритетные требования можно было отложить на более поздние выпуски, на самом деле все проекты выпускались лишь раз. Соответственно все заинтересованные лица знали, что у них есть лишь один шанс добиться реализации всей необходимой им функциональности. Поэтому всем требованиям неизменно назначался самый высокий приоритет, и их реализация каждый раз требовала недюжинных усилий от команды.

В действительности некоторые возможности системы более ценны, чем остальные. Это становится очевидным, когда на поздних стадиях требуется «быстро урезать» проект, то есть отбросить несущественные характеристики,

чтобы критически важные возможности удалось поставить в срок. В такой ситуации люди определенно принимают решения по приоритетам, но в состоянии паники. Определив приоритеты на ранних этапах разработки проекта и переоценивая их в соответствии с изменяющимися предпочтениями клиентов, условиями рынка и реалиями бизнеса, команда сможет мудро тратить свое время на создание наиболее ценных возможностей. Реализовав большую часть функции, а затем решив, что она не нужна, вы впустую потратите ресурсы и испытаете разочарование.

Предоставленные самим себе, клиенты, скорее всего, назначат 85% требований высокий приоритет, 10% — средний и 5% — низкий. Это не дает менеджеру проекта большой свободы для маневра. Если почти все требования действительно важны, вы рискуете тем, что ваш проект будет не совсем успешен. Отшлифуйте требования до блеска, чтобы отбросить не имеющие большой ценности и упростить неоправданно сложные требования. Одно исследование показало, что две трети функций, реализованных в программных системах редко или вообще не используются (The Standish Group, 2009). Чтобы представители клиентов с большим мужеством назначали требованиям низкие приоритеты, аналитику стоит задавать примерно такие вопросы:

- Есть ли другой способ удовлетворить это требование клиентов?
- Что случится, если это требование убрать или отложить?
- Что произойдет с бизнес-целями проекта, если это требование не будет реализовано в ближайшие месяцы?
- Почему пользователи будут недовольны, если реализация этого требования будет отложена до следующего выпуска?
- Стоит ли из-за этой функции откладывать выпуск всех остальных функций с тем же приоритетом?

> **Внимание!** Если вы выполнили процесс определения приоритетов и приоритеты всех требований оказались примерно одинаковыми, то на самом деле никакой приоритизации вы не выполнили.

При оценке приоритетов посмотрите на связи и взаимоотношения между требованиями и их соответствием бизнес-целям проекта. Однажды команда управления менеджментом в крупном коммерческом проекте выразила нетерпение, когда аналитик настаивал на определении приоритетов требований. Менеджеры отметили, что они могут обойтись определенной функции, но тогда в качестве компенсации придется усиливать другую функцию. Если бы они отсрочили слишком много функций, то продукт не принес бы запланированных доходов.

Конфликты возникают между заинтересованными лицами, которые убеждены, что *именно их* требования самые важные. В общем случае членам привилегированного класса пользователей должно отдаваться предпочтение при наличии требований с равноценными приоритетами. Это одна из причин выявить и оценить классы пользователей на ранних стадиях проекта.

Некоторые приемы определения приоритетов

В малом проекте заинтересованные лица могут согласовать приоритеты требований неформально. Крупные или спорные проекты со многими заинтересованными лицами требуют более структурированного подхода, устраняющего из процесса некоторые эмоции, политику и догадки. Для помощи в определении приоритетов предлагается несколько аналитических и математических методик. Они предполагают оценку относительной ценности и относительной стоимости каждого требования. Требования с самым высоким приоритетом — те, что обеспечивают большую ценность продукта при меньшей стоимости (Karlsson и Ryan, 1997; Jung, 1998). В этом разделе обсуждаются несколько приемов, используемых людьми для определения приоритетов. Чем проще, тем лучше, но только при условии, что прием эффективен.

> **Внимание!** Избегайте определения приоритетов «по децибелам», когда требование, высказанное громче всего, получает наибольший приоритет, и «по угрозе», когда заинтересованные лица, имеющие больший вес в компании, всегда получают требуемое.

Включать или не включать

Простейший метод определения приоритетов заключается в том, что группа заинтересованных лиц просматривает список требований и по каждому принимает простое бинарное решение: включать требование в следующий выпуск или нет? При этом надо держать в уме бизнес-цели проекта и пытаться свести список к абсолютному минимуму, необходимому в первом выпуске. Далее, после начала реализации этого выпуска, можно вернуться к ранее исключенным требованиям и повторить весь процесс снова для следующего выпуска.

Бабах, и нет требования!

Однажды я проводил семинар, в котором участвовали шесть заинтересованных лиц, присутствовавших лично, и еще четыре участвовали по телефону. Нам нужно было расставить по приоритетам 400 требований. Мы решили просто пройтись по списку и отобрать требования для текущего выпуска, а потом, при работе над следующими выпусками, вернуться к исключенным требованиям. Мы зарезервировали комнату для совещаний на много часов, чтобы перебрать весь список. У одного участника, топ-менеджера, было полномочие принимать окончательное решение в случае конфликтов. Вскоре после начала совещания он понял, что день будет длинным и монотонным. Он решил немного развлечь народ — каждый раз, когда мы убирали требование, он имитировал звук взрыва, как будто взрывалось требование. Процесс «прореживания» требований пошел намного веселее.

Попарное сравнение и ранжирование

Люди иногда пытаются назначать уникальные последовательные номера приоритетов всем требованиям. Ранжирование списка требований предусматривает выполнение попарного сравнения между всеми требованиями, чтобы можно было определить, какой член в каждой паре приоритетнее. Рис. 14-1 в главе 14 иллюстрирует использование таблицы для выполнения такого попарного сравнения атрибутов качества; такой же подход можно применять к функциям, пользовательским историям и другим наборам требований одного типа. Такое сравнение становится трудноуправляемым, если требований больше нескольких десятков. Оно может работать на уровне отдельных функций, но не для всех функциональных требований к системе в целом.

В реальности ранжирование всех требований по приоритетам — ненужная крайность. Вы ведь не будете реализовывать все эти требования в индивидуальных выпусках — вместо этого вы сгруппируете их в пакеты по выпускам или этапам разработки. Достаточно группировки требований по функциям или небольшим наборам требований, у которых примерно одинаковые приоритеты и которые должны реализовываться вместе.

Трехуровневая шкала приоритетов

Обычный метод расстановки приоритетов подразумевает три группы категорий требований. Неважно, как вы их назовете, все сводится к высокому, среднему и низкому приоритетам. Такие шкалы приоритетов субъективны и неточны. Лица, заинтересованные в проекте, должны согласовать, что означает каждый уровень в используемой шкале.

Один из способов оценки приоритетов предлагает учитывать два измерения: *важность* и *срочность* (Covey, 2004). Каждое требование считается важным либо не важным для достижения бизнес-целей и срочным либо не срочным. Это относительная оценка в наборе требований, а не абсолютная двоичная разница. Как показано в табл. 16-1, получаются четыре комбинации для определения шкалы приоритетов:

- требования *с высоким приоритетом* (high priority) — и важные (пользователям нужны функции), и срочные (они необходимы уже в следующем выпуске). Некоторые требования приходится включать в эту категорию согласно контрактным или юридическим обязательствам либо из-за непреодолимых бизнес-причин. Если реализацию требования можно отложить до более позднего выпуска без отрицательных последствий, тогда, в соответствии с этим определением, его нельзя считать высокоприоритетным;
- требования со *средним приоритетом* (*medium priority)* — важные (пользователям нужны функции), но не срочные (они могут ждать следующего выпуска);
- требования с *низким приоритетом* (*low priority)* — не важные (пользователи при необходимости могут обойтись без этой функций) и не срочные (пользователи могут ждать, возможно неограниченно долго);

- требования в четвертой клетке кажутся срочными части заинтересованных лиц, возможно по каким-то «политическим» причинам, но в действительности они не важны для достижения бизнес-целей. Не тратьте время на работу над ними, потому что они не добавят продукту ценности. Если они не важны, присвойте им низкий приоритет или вообще удалите.

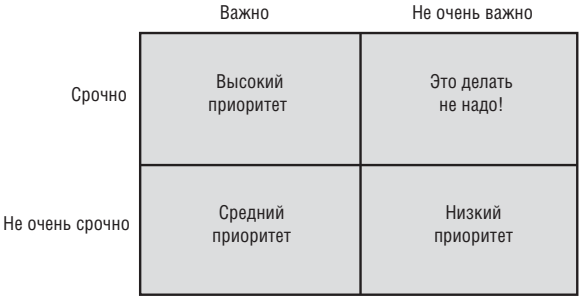


Рис. 16-1. Определение приоритетов требований по важности и срочности

Включайте приоритеты каждого требования как атрибут в описания вариантов использования, спецификацию требований или базу данных требований. Установите соглашения, чтобы пользователь знал, присваивается ли приоритет, назначенный высокоуровневому требованию, всем дочерним требованиям или же каждое функциональное требование должно иметь свой собственный атрибут приоритетности.

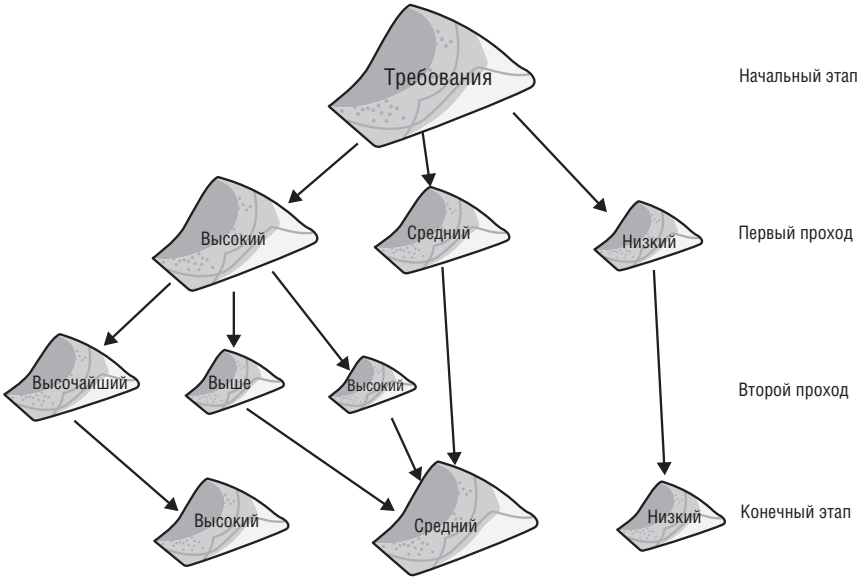


Рис. 16-2. Многопроходная приоритизация обеспечивает ориентацию на получение набора высокоприоритетных требований

Иногда, особенно в крупных проектах может потребоваться определить приоритеты итеративно. Попросите членов команды разбить требования на

три группы по приоритету — высокий, средний или низкий. Если число высокоприоритетных требований слишком большое и вы не уверены, что они все действительно *должны* присутствовать в следующем выпуске, сделайте следующий проход разбиения всех таких требований на три группы. Если хотите можно назначить им приоритеты «высокий», «выше» и «высочайший», чтобы участники не забыли, что эти требования изначально являются важными. Требования высочайшего приоритета становятся новой группой высокоприоритетных требований. Переместите требованиям с приоритетами «высокий» и «выше» в исходную среднеприоритетную группу (рис. 16-2). Жесткий подход к критерию «должно быть в следующем выпуске или он вообще не будет выпущен» помогает команде сосредоточиться на действительно высокоприоритетных функциях.

При выполнении анализа приоритетов по трехуровневой шкале нужно помнить о зависимостях требований. У вас будут проблемы, если высокоприоритетное требование зависит от другого, у которого приоритет ниже и которое запланировано к реализации позже.

Схема классификации приоритетов MoSCoW

Четыре заглавных буквы в схеме определения приоритетов MoSCoW обозначают четыре возможных классификации приоритетов в наборе (IIBA, 2009):

- **M**ust — требование должно быть удовлетворено, чтобы решение было успешным;
- **Should** — требование важно и по возможности должно быть включено в решение, но оно не является условием успеха решения;
- **Could** — это желательная функция, но ее можно отложить или удалить. Реализуйте ее, только если позволяет время и ресурсы;
- **Won't** — так обозначается требование, которое в этот раз реализовываться не будет, но может быть включено в будущий выпуск.

Схема MoSCoW заменяет трехуровневую шкалу — высокий, средний и низкий приоритет — на четырехуровневую. Она не предлагает никаких критериев принятия решения об относительном приоритете одного требования по отношению к другим. Схема MoSCoW неоднозначна в том, что касается временны*х* характеристик, особенно когда речь идет о рейтинге «Won't» — это может означать «не в следующем выпуске» или «никогда». Такие различия должны определяться четко, чтобы у всех заинтересованных лиц было единое представление о последствиях данного конкретного рейтинга. Описанная ранее трехуровневая шкала, которая базируется на анализе двумерных измерений важности и срочности и особо фокусируется на грядущем выпуске или периоде разработки, более точна в отношении приоритетов. Мы не рекомендуем пользоваться MoSCoW.

MoSCoW на практике

Один консультант описал, как его компания-клиент использовала на практике схему MoSCoW в своих проектах. «Все действо происходит вокруг рейтинга «M» практически для каждой выявленной функции и требования, — сказал он. — Если чему-то не назначен приоритет «M», это практически наверняка не будет построено. Хотя изначальное намерение заключалось в определении приоритетов, пользователи давно поняли, что никогда не нужно предоставлять ничего, что не содержит «M». Видят ли они тонкие различия между «S», «C» и «W»? Понятия не имею. Но они поняли последствия такой ранжировки. Они их не различают и понимают их всех как «в обозримом будущем этого не будет».

100 долларов

Определение приоритетов подразумевает выделение ограниченных ресурсов для достижения максимальной выгоды от средств, которые организация потратила на проект. Один из способов сделать расстановку приоритетов более наглядной — перевести ее в область более понятных ресурсов, то есть денег. В данном случае речь идет о мнимых деньгах, но все равно это деньги.

Дайте команде, занимающейся определением приоритетов сто воображаемых долларов. Члены команды должны потратить эти доллары на «покупку» конкретных требований из всего набора требований-кандидатов. На более приоритетные требования надо выделить больше денег. Если одно требование в три раза важнее для заинтересованного лица, чем другое, оно должно выделить, скажем, десять долларов на первое и три — на второе. Но каждый получает только 100 долларов — когда деньги кончаются, ничего больше нельзя реализовать, по крайней мере не в том выпуске, над которым сейчас идет работа. Один способ заключается в привлечении к процессу приоритизации различных участников для распределения своих долларов, после чего суммируются доллары по каждому требованию, чтобы определить, за какие требования было отдано больше всего долларов.

Стодолларовый подход — неплохой способ заставить группу людей мыслить в терминах выделения ресурсов на основе приоритета. Но Дейвис (Davis, 2005) указывает на несколько способов «заиграть» процесс в попытке сместить результаты в нужную сторону. Например, если вы *действительно* хотите какое-то требование, вы можете попытать отдать ему все свои сто долларов, чтобы поднять его в первые строчки списка. Но в реальности вы никогда не примете систему, в которой реализовано только это одно требование. Также эта схема не учитывает относительных усилий, необходимых для реализации каждого из этих требований. Если бы вы могли получить три требования по 10 долларов каждое с теми же усилиями, как одно за 15 долларов, вы скорее всего предпочтете взять три. Эта схема основывается исключительно на воспринимаемой ценности конкретных требований для опреде-

ленного набора заинтересованных лиц, а это является ограничением многих методик определения приоритетов.

Еще одна методика приоритизации основана на реальных, а не воображаемых деньгах. В методике *объективной цепочки* (objective chain) Джоя Битти и Энтони Чена (Joy Beatty и Anthony Chen, 2012) вам приходится назначать примерную долларовую цену, которая представляет, сколько каждая предлагаемая функция вносит в достижение бизнес-целей проекта. После этого можно сравнить относительную ценность функций между собой и отобрать те, что будут реализовываться в первую очередь.

Определение приоритетов на основе ценности, стоимости и риска

Когда заинтересованные лица не могут достичь соглашения о приоритетах требований при использовании других сравнительно неформальных методик, может оказаться полезным применить более аналитический метод. Прием, который называется Quality Function Deployment, или QFD, — решающий, всесторонний метод определения относительной ценности для клиента предлагаемых функций продукта (Cohen, 1995). Тем не менее, по-видимому, лишь немногие организации-разработчики ПО готовы последовательно применять QFD, несмотря на то, что структурированный метод определения приоритетов на основе QFD доказал свою полезность.

В табл. 16-1 показана табличная модель, которая поможет вам оценить относительные приоритеты для набора вариантов требований. Эта методика была признана самой эффективной в сравнительной оценке 17 методик определения приоритетов требований (Kukreja и другие, 2012). В прилагаемом к этой книге материале есть соответствующая электронная таблица Microsoft Excel. В табл. 16-1 перечислено несколько функций Chemical Tracking System (а каких же еще?). Эта схема заимствует из QFD концепцию обоснования ценности для пользователя с учетом как *выгоды* для пользователя, если функция реализована в продукте, так и *неудобства*, если она отсутствует (Pardee, 1996). Привлекательность функции прямо пропорциональна ее полезному действию и обратно пропорциональна *стоимости* и *риску*, связанному с ее реализацией. При прочих равных условиях функции с наибольшим значением соотношения «ценность/стоимость» должны иметь наивысший приоритет. При этом набор оцениваемых приоритетов распределяется по всему континууму, а не группируется лишь по нескольким дискретным уровням.

Табл. 16-1. Пример матрицы определения приоритетов для Chemical Tracking System

Относительный вес		**2**	**1**			**1**		**0,5**		
Функция		**Относительная выгода**	**Относительный ущерб**	**Общая ценность**	**Ценность, %**	**Относительная стоимость**	**Стоимость, %**	**Относительный риск**	**Риск, %**	**Приоритет**
1.	Распечатка списка данных по безопасности материалов	2	4	8	5,2	1	2,7	1	3,0	1,22
2.	Запрос о статусе заказа поставщика	5	3	13	8,4	2	5,4	1	3,0	1,21
3.	Создание отчета о инвентаризации склада химикатов	9	7	25	16,1	5	13,5	3	9,1	0,89
4.	Просмотр истории конкретного контейнера с химикатом	5	5	15	9,7	3	8,1	2	6,1	0,87
5.	Поиск химиката по каталогам поставщиков	9	8	26	16,8	3	8,1	8	24,2	0,83
6.	Поддержка списка опасных химикатов	3	9	15	9,7	3	8,1	4	12,1	0,68
7.	Изменение текущих заявок на химикаты	4	3	11	7,1	3	8,1	2	6,1	0,64
8.	Создание отчетов об инвентаризации отдельных лабораторий	6	2	14	9,0	4	10,8	3	9,1	0,59
9.	Проверка в базе данных по обучению наличия записи о прохождении обучения по обращению с опасными веществами	3	4	10	6,5	4	10,8	2	6,1	0,47

Табл. 16-1. (*окончание*)

Относительный вес	**2**	**1**			**1**		**0,5**		
Функция	**Относительная выгода**	**Относительный ущерб**	**Общая ценность**	**Ценность, %**	**Относительная стоимость**	**Стоимость, %**	**Относительный риск**	**Риск, %**	**Приоритет**
10. Импорт химических структур из инструментальных средств для рисования структур	7	4	18	11,6	9	24,3	7	21,2	0,33
Итоги	**53**	**49**	**155**	**100,0**	**37**	**100,0**	**33**	**100,0**	

Примените эту схему определения приоритетов к независимым требованиям, то есть заведомо не имеющим наивысшей ценности. В частности, не нужно включать в этот анализ элементы, которые реализуют основные бизнес-функции продукта, которые вы считаете основными отличительными чертами продукта или необходимые для выполнения норм закона. Определив функции, которыми обязательно должен обладать выпускаемый продукт, используйте табл. 16-1, чтобы оценить по шкале относительных приоритетов оставшиеся возможности. Обычно в процессе назначения приоритетов участвуют:

- менеджер проекта или бизнес-аналитик, который ведет процесс, разрешает конфликты и при необходимости корректирует данные о приоритетах, поступающие от других участников;
- представители клиента — сторонники продукта, маркетологи или владелец продукта, предоставляющие информацию о выгоде и рисках продукта;
- представители разработчиков, предоставляющие данные о стоимости и риске.

При использовании этой модели определения приоритетов пользуйтесь следующим планом:

1. Перечислите в таблице все функции, варианты использования, направления вариантов использования, пользовательские истории или функциональные требования, для которых хотите определить относительные приоритеты; для примера мы взяли функции. Все элементы должны принадлежать к одному уровню абстракции — не смешивайте функциональные требования с функциями продукта, вариантами использования или пользовательскими историями. Если некоторые функции логически связаны (например, вы реализуете функцию B только при наличии функции A) или взаимозависимы, в анализ включайте только ведущую функцию. Эта модель работает только для нескольких десятков функций, при большем числе она становится слишком громоздкой. Если у вас элементов больше нескольких десятков, сгруппируйте связанные функции, чтобы составить

управляемый начальный список. Этот метод можно применять в рамках иерархии. После первичного определения приоритетов, скажем, функций, можно повторить процедуру внутри функции, чтобы задать приоритеты отдельных подфункций или функциональных требований.

2. Попросите представителей клиентов оценить относительную выгоду, которую каждая функция дает клиенту или бизнесу, по шкале от 1 до 9: 1 балл означает, что никто не находит ее полезной, а 9 — что она крайне ценная. Эти оценки выгоды отражают соответствие функций бизнес-требованиям к продукту.
3. Оцените относительный ущерб, который понесет клиент или бизнес, если функция *не будет* реализована. Снова используйте шкалу от 1 до 9: 1 балл означает, что никто не расстроится, если она будет исключена; 9 показывает серьезный ущерб. Требования, имеющие низкие баллы и выгоды, и урона, увеличивают стоимость, но приносят мало пользы. Иногда у функции достаточно низкая ценность, если ее использовать будет немного клиентов, но большой ущерб, если конкурирующий продукт может похвастаться этой функцией и клиенты ожидают, что она должна быть — даже если сами лично не планируют ее использовать! Маркетологи иногда называют эти функции «обязательным пунктом»: вы должны сказать, что она у вас есть, даже если это интересно очень немногим. Оценивая уровень ущерба, учитывайте, подумайте, что может произойти, если вы не включите функцию:
   - Проиграет ли ваш продукт по сравнению с другими, в которых эта функция есть?
   - Будут ли какие-либо юридические или контрактные последствия?
   - Не нарушите ли вы какую-либо норму права или отраслевой стандарт?
   - Не лишит ли это пользователей возможности выполнять какие-либо необходимые или ожидаемые действия?
   - Намного ли сложнее добавить эту функцию позже, в качестве усовершенствования?
   - Возникнут ли проблемы из-за того, что отдел маркетинга уже обещал эту функцию, некоторым клиентам?
4. В этой таблице подсчитаны итоговые значения для каждой функции как сумма баллов ее выгоды и ущерба (распределение весов описывается далее в этой главе). В ней суммируется ценность всех функций и вычисляется процент от общего значения для каждого набора функций, равного сумме ценностей каждой функции (столбец «Ценность, %»). Заметьте, что это не процентная доля общей суммы для всего продукта, а только для набора функций, для которых вы определяете приоритеты друг относительно друга.
5. Попросите разработчиков оценить относительную стоимость реализации каждой функции по шкале от 1 (легко и быстро) до 9 (трудоемко и дорого).

В таблице подсчитан процент общей стоимости, составленной из вклада каждой функции. Разработчики оценивают рейтинги стоимости на основе сложности функции, объема требуемой работы над пользовательским интерфейсом, потенциальную возможность повторного использования существующего кода, объем необходимого тестирования и документации и т. п. В командах гибкой разработки эти оценки могут основываться на баллах, назначенных пользовательским историям. (Подробнее об оценке в проектах гибкой разработки см. главу 19.)

6. Подобным же образом, разработчики оценивают относительную степень технического риска (не бизнес-риска), связанного с каждой функцией, по шкале от 1 до 9. Технический риск — это вероятность *неудачной* реализации функции с первого раза. 1 балл означает, что вы сможете запрограммировать ее даже во сне, 9 баллов означает серьезную озабоченность возможностью реализации, нехваткой сотрудников с необходимым опытом или использованием неиспытанных или незнакомых средств и технологий. В таблице подсчитан общий процент риска, который складывается из риска каждой функции.
7. После того как вы введете все результаты оценки в таблицу, по следующей формуле подсчитывается значение приоритета для каждой функции:

$$\text{приоритет} = \frac{\text{ценность \%}}{\text{затраты \%} + \text{риск \%}}$$

8. Наконец, отсортируйте список функций по уменьшению подсчитанного приоритета — крайний правый столбец. Функции вверху списка характеризуются наиболее благоприятным сочетанием ценности, стоимости и риска, и поэтому — при равенстве остальных факторов — должны иметь наивысший приоритет. Целевое обсуждение функций, оказавшихся вверху списка позволят вам уточнить эту предварительную оценку и получить последовательность приоритетов, с которой могут согласиться заинтересованные лица, несмотря на то, что никто не получает ровно того, что хотел.

По умолчанию веса выгоды и ущерба одинаковы. Можно изменить относительные веса четырех множителей вверху таблицы для отражения принятого в вашей команде процесса обдумывания решений по приоритетам. В табл. 16-1 вес всех баллов выгоды в два раза больше баллов ущерба, веса ущерба и затрат одинаковы, а вес риска составляет половину веса ущерба или затрат. Чтобы исключить какой-то показатель из модели, присвойте ему нулевой вес.

При использовании табличной модели для работы с участниками вам скорее всего потребуется скрыть некоторые столбцы табл. 16-1: «Общая ценность», «Ценность, %», «Стоимость, %» и «Риск %». Они содержат промежуточные результаты вычислений, которые могут только отвлекать. Скрыв их, вы позволите клиентам сосредоточиться на четырех категориях рейтингов и вычисленных значений приоритетов.

А можно еще на кулачках

Одна компания, которая ввела у себя процедуру определения приоритетов, основанную на таблице, показанной в этой главе, обнаружила, что это помогло команде, работающей над проектом, выйти из тупика. Несколько заинтересованных в проекте лиц придерживались различных мнений о том, какие функции наиболее важны для проекта, и команда оказалась в безвыходном положении. Анализ с помощью таблицы сделал оценку приоритетов более объективной и менее эмоционально накаленной, позволив команде прийти к некоторым удовлетворительным заключениям и двигаться дальше.

Консультант Джоанна Ротман (Johanna Rothman, 2000) писала: «Я предлагала эту электронную таблицу своим клиентам в качестве инструмента для принятия решений. Хотя те, кто пробовал ей пользоваться, ни разу не заполнили ее до конца, они обнаружили, что дискуссия, которая начиналась при ее использовании, чрезвычайно полезна для определения относительных приоритетов различных требований». Таким образом, каркас концепции выгоды, ущерба, стоимости и риска можно применять для направления дискуссий о приоритетах. Это более ценно, чем формально выполнить полную проработку анализа электронной таблицы и затем полагаться исключительно на вычисленную последовательность приоритетов. Требования и их приоритеты со временем могут меняться, поэтому таблицу можно применять на всем протяжении проекта для управления оставшейся работой.

Точность этого метода ограничена возможностью команды оценивать выгоду, ущерб, стоимость и риск для каждого элемента. Поэтому используйте подсчитанную последовательность приоритетов только как ориентир. Заинтересованные лица должны проверить итоговую таблицу, чтобы прийти к соглашению о рейтингах и результирующей последовательности приоритетов. Если вы не уверены, можете ли доверять результатам, откалибруйте эту модель под себя, используя набор готовых требований из какого-нибудь предыдущего проекта. Скорректируйте значения весов так, чтобы подсчитанная последовательность приоритетов хорошо сочеталась с вашей постфактум-оценкой того, насколько в действительности важны требования в вашем калибровочном наборе. Это даст вам некоторую уверенность в использовании данной таблицы в качестве модели для расстановки приоритетов в своих проектах.

> **Внимание!** Не придавайте слишком большого значения малым отличиям в подсчитанных значениях приоритетов. Данный полуколичественный метод не претендует на математическую точность. Ищите группы требований с похожими значениями приоритетов.

Различные заинтересованные в проекте лица часто имеют конфликтующие мнения о важности того или иного требования и потенциальном уроне

от его отсутствия. Один из вариантов таблицы приоритетов учитывает входные данные от нескольких классов пользователей или других заинтересованных в проекте лиц. На странице Multiple Stakeholders (Множество пользователей) электронной таблицы из состава прилагаемых к этой книге материалов скопируйте столбцы Relative Benefit (Относительная выгода) и Relative Penalty (Относительный урон) для каждого заинтересованного в проекте лица, участвующего в анализе. Затем назначьте вес для каждого участника, больший вес отдавая привилегированным классам пользователей, а не группам, имеющим меньшее влияние на принятие решений, касающихся проекта. В таблице будет учтен вес каждого заинтересованного лица при подсчете общего значения ценности.

Эта модель также поможет вам принимать решения о компромиссах при оценке предлагаемых изменений в требованиях. Добавляйте новые требования в таблицу приоритетов и наблюдайте, как их приоритеты соответствуют приоритетам требований в принятой базовой версии, чтобы выбирать наилучшую последовательность реализации.

Не обязательно каждый раз использовать такой сложный метод. Старайтесь, чтобы ваш процесс определения приоритетов был насколько возможно простым, но не более того. Стремитесь превратить обсуждение требований из политической баталии в форум, где заинтересованные в проекте лица смогут давать честные оценки. Это даст вам лучшую возможность создавать продукты, обладающие максимальной бизнес-ценностью.

Что дальше?

- Переоцените требования в резерве своего текущего проекта, используя определения на рис. 16-1, чтобы отделить требования, которые действительно должны быть включены в следующий выпуск, от тех, которые при необходимости могут подождать. Изменило ли это какие-либо из ваших приоритетов?
- Примените модель определения приоритетов, описанную в этой главе, к 10-15 функциям, вариантам использования или пользовательским историям из недавнего проекта. Насколько хорошо подсчитанные этим способом приоритеты совпадают с теми, которые вы определили каким-либо другим методом? Насколько хорошо они соответствуют вашим субъективным ощущениям правильной расстановки приоритетов?
- Если обнаружилась диспропорция между приоритетами, предсказанными моделью, и теми, которые кажутся вам правильными, проанализируйте, какая часть модели не дает ожидаемых результатов. Попытайтесь использовать разные значения веса для выгоды, ущерба, стоимости и риска. Корректируйте модель до тех пор, пока она не даст результаты, соответствующие вашим ожиданиям, иначе нельзя будет доверять ее возможностям прогнозирования.

- Откалибровав модель расстановки приоритетов, примените ее к новому проекту. Включите подсчитанные приоритеты в процесс принятия решений. Посмотрите, дает ли это результаты, которые заинтересованным в проекте лицам кажутся более удовлетворительными, чем те, что получались при прежнем подходе.
- Попробуйте сегодня воспользоваться методикой расстановки приоритетов, которую вы раньше никогда не использовали. Например, если вы уже применяете MoSCoW, попробуйте воспользоваться трехуровневой шкалой и сравнить результаты.

Глава 17

Утверждение требований

Барри, ведущий тестировщик, вел семинар, участники которого проверяли спецификацию требований к ПО на наличие проблем. Во встрече принимали участие представители двух классов пользователей — разработчик Джереми и бизнес-аналитик Триш, которая написала спецификацию требований к ПО. В одном требовании было указано: «Система должна обеспечить отключение неактивных терминалов системы обучения по истечении тайм-аута». Джереми представил группе свою интерпретацию этого требования: «В этом требовании говорится, что система автоматически отключит пользователя любой рабочей станции, подключенной к системе обучения, если на ней наблюдается бездействие в течение определенного периода времени».

Первым высказался Хью-Ли, один из сторонников продукта. «Как система определяет, что терминал не активен? Так же, как включается заставка на мониторе: если в течение, скажем, пяти минут мышь или клавиатура остаются в покое, то сеанс пользователя закрывается? Это может раздражать, например, когда пользователь просто отвлекся на разговор».

Триш добавила: «На самом деле в требовании ничего не говорится о завершении сеанса пользователя. Я предположила, что это означает окончание сеанса, но, может, достаточно просто ввести пароль».

Джереми был также озадачен. «В этом требовании речь идет о любой рабочей станции, которая может подключиться к системе обучения или только о рабочих станциях, которые активны и подключены к системе обучения в данный момент? О какой продолжительности времени ожидания мы говорим? Возможно, существует некое руководство по безопасности для таких случаев?»

Барри убедился, что секретарь точно зафиксировал все точки зрения. «Ну хорошо, похоже, что в требовании есть несколько неясностей и не хватает некоторой информации, в частности, не определено время ожидания. Триш, не могла ли бы ты связаться с координатором отдела безопасности и решить этот вопрос?».

Большинству разработчиков знакомо неприятное чувство, возникающее, если их просят реализовать слишком неясные или неполные требования. Если они не могут получить необходимую информацию, они вынуждены

ориентироваться на собственные предположения, которые не всегда верны. Как мы видели в главе 1, потребуется много усилий, чтобы исправить ошибки в требованиях, работа над которыми уже завершена. Другие исследования свидетельствуют, что на исправление ошибки, выявленной на стадии работы над требованиями, тратится в среднем 30 минут, тогда как на исправление ошибки, выявленной в ходе тестирования системы, необходимо от 5 до 17 часов (Kelly, Sherif и Hops, 1992). Ясно, что любые усилия, затраченные на выявление ошибок в спецификации к требованиям, сэкономят реальные время и деньги.

Во многих проектах тестирование — одна из последних стадий проекта. Проблемы, связанные с требованиями, могут оставаться в продукте до тех пор, пока они не будут выявлены в ходе долгого тестирования системы или — хуже всего — пока их не обнаружит клиент. Если вы начнете планирование тестирования и разработку тестов на ранней стадии проекта, вы сможете обнаружить многие ошибки вскоре после их появления. При этом вы предотвратите ущерб, который они могу нанести в будущем, и снизите затраты на тестирование и обслуживание.

На рис. 17-1 показана V-образная модель разработки ПО, когда тестирование выполняется параллельно с разработкой. Эта модель указывает, что приемочные тесты основываются на пользовательских требованиях, тесты системы — на функциональных требованиях, а тесты целостности — на архитектуре системы. Эта модель применима независимо от того, разработка какой версии ПО тестируется — продукта в целом, отдельного выпуска или отдельного инкремента.

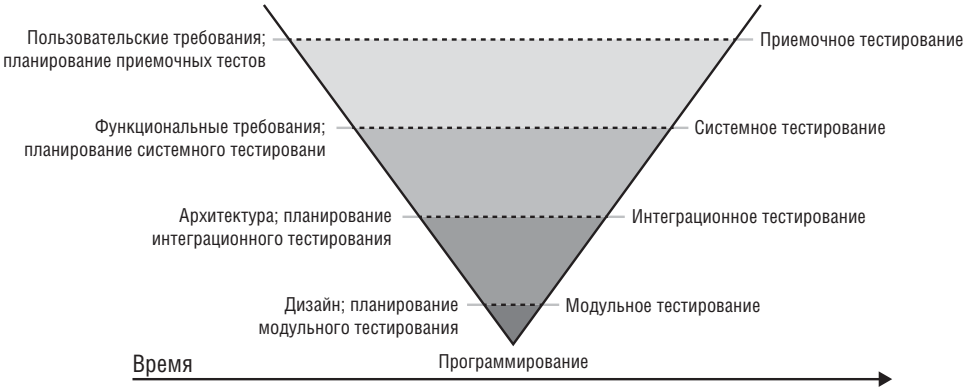


Рис. 17-1. V-образная модель разработки ПО включает планирование и проектирование тестирования на ранних этапах

Как будет сказано далее в этой главе, тесты можно использовать для проверки требований каждого из этих типов во время разработки. В процессе разработки требований вы не можете проводить никаких тестов, так как никакого ПО еще нет. Однако концептуальные (то есть, не зависящие от реализации) тесты, созданные на основе требований, позволят выявить ошибки,

неясности и пропуски данных в спецификации требований к ПО и проанализировать модели задолго до того, как разработчики приступят к написанию кода.

Участники проекта иногда с неохотой тратят время на проверку и тестирование спецификации требований к ПО. Им кажется, что если выделить время на улучшение качества требований, дата выпуска продукта задержится на такой же срок. Однако при таком отношении вы можете смело ожидать нулевых результатов от всех усилий, затраченных на утверждение требований. В действительности же эти усилия могут *сократить* график поставки, за счет сокращения числа требуемых исправлений и ускорения интеграции и тестирования системы (Blackburn, Scudder и Van Wassenhove, 1996). Чем лучше требования, тем выше качество продукта и тем более доволен клиент, что в свою очередь снизит затраты на обслуживание, улучшение и клиентскую поддержку продукта. Инвестируя в качество продукта, вы всегда сэкономите больше, чем потратили.

Для оценки корректности и качества требований можно применять различные приемы (Wallace и Ippolito, 1997). Один из таких приемов заключается в выражении требования в количественном представлении, с тем чтобы вы смогли продумать способ измерения того, насколько хорошо предложенное решение. Сьюзан и Джеймс Роберсоны (Robertson и Robertson, 2013) используют термин *критерий годности* (fit criteria) для подобных приемов. В этой главе рассматриваются приемы проверки официального и неофициального рецензирования требований, разработки тестов на основании требований и определения клиентом критериев приемлемости продукта.

Утверждение и верификация

Утверждение (validation) требований является четвертым этапом — наряду со сбором информации, анализом и созданием спецификации — разработки требований. Для описания этого этапа некоторые авторы используют термины *проверка* или *верификация* (Thayer и Dorfman, 1997). В этой книге мы применяем терминологию из книги «Software Engineering Body of Knowledge» (Abran и другие, 2004) и называем четвертую составляющую разработки требования утверждением. Утверждение требований применяется, чтобы убедиться, что у них есть все требуемые свойства высококачественных требований, также необходимое мероприятие. Если быть точным, то в применении к разработке ПО утверждение и верификация — два разных действия. *Верификация* позволяет определить, соответствует ли требованиям продукт ряда операций по разработке (выполняет правильно определенную вещь). *Утверждение* требований позволяет удостовериться в том, что продукт удовлетворяет потребности клиентов (делает все правильно).

Если распространить эти определения на требования, то получается, что верификация позволяет определить, правильно ли написаны требования, то есть у ваших требований есть требуемые свойства, описанные в главе 11.

Утверждение требований позволяет выяснить, написали ли вы правильные требования, то есть что они соответствуют бизнес-целям. Эти два понятия тесно связаны. Для простоты в этой главе мы говорим об утверждении требований, но описываемые здесь приемы полезны для получения как правильных, так и высококачественных требований.

Утверждение требований позволяет командам создавать правильное решение, соответствующее заявленным бизнес-целям. Утверждение требований позволяет удостовериться в том, что:

- в спецификации требований к ПО должным образом описаны предполагаемые возможности и характеристики системы, которые удовлетворят потребности различных заинтересованных в проекте лиц;
- требования к ПО точно отражают бизнес-требования, системные требования, бизнес-правила и другие источники;
- требования полные, реализуемые и проверяемые;
- все требования необходимы, а всей их совокупности достаточно для достижения бизнес-целей;
- все требования согласованы друг с другом;
- требования обеспечивают качественную основу для дизайна и создания ПО.

Утверждение — это не один отдельный этап процесса, выполняемый после выявления и документирования требований. Некоторые проверочные мероприятия, например просмотр все более разрастающейся спецификации требований к ПО, выполняются после каждой процедуры выявления требований, их анализа и документирования. Другие мероприятия, такие, как формальная проверка спецификации требований к ПО, обеспечивают создание базового набора требований. Включите в план проекта этапы утверждения требований в качестве отдельных задач. Разумеется, вы можете утвердить только задокументированные требования, а не предполагаемые, которые существуют только в чьем-то воображении.

Рецензирование требований

Всякое исследование продукта ПО на предмет выявления проблем любым другим лицом, кроме его автора, называется *рецензированием* (peer review). Просмотр задокументированных требований — эффективный прием выявления неясных или не поддающихся проверке требований, требований которые были определены недостаточно ясно для разработки, а также других проблем.

Различные типы рецензирования называются по-разному (Wiegers, 2002). Неофициальное рецензирование полезны при знакомстве людей с продуктом и сборе неструктурированных отзывов о нем. Однако они несистематические, неполные и несогласованные. Есть несколько видов неофициального рецензирования требований:

- *проверка «за столом»* (peer deskcheck), когда вы просите одного коллегу исследовать ваш продукт;
- *коллективная проверка* (passaround), когда вы приглашаете несколько коллег для параллельной проверки продукта;
- *сквозной разбор* (walkthrough), когда автор описывает создаваемый продукт и просит его прокомментировать.

Неформальные рецензии хороши для выявления явных ошибок, противоречий и пробелов. Они могут помочь обнаружить формулировки, не соответствующие характеристикам высококачественных требований. Но рецензенту сложно в одиночку обнаружить все неоднозначные требования. Он может прочитать требование и подумать, что понимает его, и перейти к следующему без какой-либо задней мысли. Прочитав это же требование, другой рецензент может интерпретировать его иначе, но также не найти проблемы. Если эти два рецензента не обсудят требование, неоднозначность может остаться незамеченной и проявиться только на поздних этапах проекта.

Формальное рецензирование представляет собой строго регламентированный процесс. По его завершении формируется отчет, в котором указаны материал, рецензенты и мнение команды рецензентов о приемлемости продукта. Главный результат — совокупность всех найденных дефектов и поднятых вопросов. Члены команды, которая выполняет официальную рецензию продукта, совместно отвечают за качество рецензии, хотя в конце концов за качество продукта отвечают все-таки его создатели.

Наиболее хорошо себя зарекомендовавшая себя форма официального рецензирования *экспертиза* (inspection). Наверное, экспертизу документации требований — наиболее действенный из доступных приемов проверки качества ПО. Несколько компаний сэкономили ни много ни мало — целых 10 часов труда на каждый час, потраченный на экспертизу документации требований и других готовых к поставке продуктов (Grady и Van Slack, 1994). Десятикратная выгода от экспертизы не то, от чего можно было бы просто отмахнуться.

Если вы серьезно решили совершенствовать качество вашего ПО, то ваша команда должна обследовать каждый написанный документ, касающийся требований. Детальная проверка объемной документации может быть скучной и долгой. Однако все мои знакомые, кто занимался экспертизой требований, соглашаются, что каждая минута была потрачена не зря. Если у вас не хватает времени на проверку каждой детали, воспользуйтесь методами анализа рисков, чтобы отобрать требования, нуждающиеся в экспертизе, от тех, которые менее критичны, сложны, или новы и для проверки которых достаточно неофициального рецензирования. Экспертизы стоят недешево. И не очень приятны. Но они дешевле — и веселее, чем альтернативный вариант, заключающийся в затратах огромных сил и риске потерять расположение клиентов из-за слишком позднего обнаружения и устранения проблем.

Подойдешь поближе — увидишь побольше

В Chemical Tracking System представители пользователей проводили неофициальное рецензирование последней версии спецификации требований после каждого семинара, посвященного выявлению требований. Таким образом удавалось выявить множество ошибок. После завершения выявления требований, один аналитик свел всю информацию, полученную от всех классов пользователей в одну спецификацию требований к ПО — 50 страниц плюс несколько приложений. Затем два бизнес-аналитика, один разработчик, три сторонника продукта, менеджер проекта и один тестировщик обследовали этот документ за три двухчасовых совещания, проведенных в течение недели. Они дополнительно обнаружили 233 ошибки, в том числе десятки серьезных дефектов. Все проверяющие согласились, что время, потраченное на «прочесывание» спецификации (по одному требованию за раз) сэкономило несчетное количество часов в дальнейшем.

Процесс экспертизы

Процесс экспертизы был разработан Майклом Фэганом (Fagan, 1976; Radice, 2002), а другие дополнили и модифицировали его методы (Gilb и Graham, 1993; Wiegers, 2002). Экспертиза была признана рекомендуемым приемом в области разработки ПО (Brown, 1996). Этот метод годится для проверки любого продукта, в том числе документации требований и дизайна, исходного кода, документации тестирования и планов проекта.

Экспертиза это четко определенный многоэтапный процесс. В ней участвует небольшая команда участников, которые тщательно проверяют продукт на наличие дефектов и изучают возможности улучшения. Экспертизы обеспечивают уровень качества продукта, предшествующий окончательному. Существует несколько форм экспертизы, но все это мощные методы обеспечения качества. Дальнейшее описание базируется на методике экспертизы Фэгана.

Участники

До начала экспертизы позаботьтесь о привлечении всех необходимых людей — в противном случае может оказаться, что после устранения проблем найдется кто-то важный, кто не согласится с изменениями. Участники экспертизы должны отражать четыре точки зрения на продукт (Wiegers, 2002):

- **Автор продукта и, возможно, коллеги автора** Это точка зрения бизнес-аналитика, составившего документацию требований. Если возможно, включите еще одного опытного бизнес-аналитика, потому что он знает, за какими ошибками написания требований нужно следить.
- **Люди, послужившие источником информации для элемента, который следует проверять** В этой роли могут выступить представители пользова-

телей или автор предыдущей спецификации. При отсутствии спецификации более высокого уровня, в экспертизе должны принять участие представители клиента, например сторонники продукта, чтобы подтвердить, что требования в спецификации требований к ПО описаны правильно и полно.

- **Люди, которые будут выполнять работу, основанную на проверяемом элементе** Для экспертизы спецификации требований к ПО вы можете привлечь разработчика, тестировщика, менеджера проекта, а также составителя пользовательской документации, потому что они смогут выявить проблемы различных типов. Тестировщик скорее всего выявит требования, не поддающиеся проверке, а разработчик сумеет определить требования, которые технически невыполнимы.
- **Люди, отвечающие за работу продуктов, взаимодействующих с проверяемым элементом.** Они выявят проблемы с требованиями к внешнему интерфейсу. Они также могут выявить требования, изменение которых в проверяемой спецификации повлияет на другие системы (эффект волны).

Постарайтесь ограничить команду семью членами или меньше. Это означает, что некоторые точки зрения не будут представлены при каждой проверке. Большие команды могут легко увязнуть в посторонних дискуссиях, попытках решения проблем и дебатов о том, что считать ошибкой. При этом снижается темп проверки и растет стоимость обнаружения каждого дефекта.

Обычно менеджер автора не должен посещать посвященное экспертизе совещание, если только он не вносит активный вклад в проект и его присутствие приемлемо для автора. Эффективная экспертиза, обнаружившая много дефектов, может создать у излишне критичного менеджера плохое впечатление об авторе. Кроме того, присутствие менеджера может подавить инициативу со стороны других участников.

Роли экспертов

Все участники экспертизы, включая автора, ищут недостатки и возможности улучшения. Участники при этом играют различные роли (Wiegers, 2002).

Автор Он создает или поддерживает проверяемый продукт. В роли автора спецификации требований к ПО, как правило, выступает бизнес-аналитик, который осуществлял выявление требований клиента и писал спецификацию. В ходе неофициальных проверок, таких, как сквозной разбор, именно автор часто возглавляет дискуссию. Однако в экспертизе он принимает более пассивное участие. Он не должен брать на себя обязанности других участников — координатора, читателя или секретаря. Поскольку автор не принимает активное участие, он имеет возможность выслушать других участников экспертизы, отвечать — но не вступать в дебаты — на их вопросы и думать. Так автору часто удается обнаружить ошибки, не замеченные другими участниками.

Координатор. Координатор, или модератор, планирует экспертизу совместно с автором, согласовывает действия и организует совещание. Координатор

распределяет проверяемый материал, а также все необходимые предшествующие документы, между участниками за несколько дней до совещания. Он отвечает за своевременное начало совещаний, поощрение вклада каждого участника и управление дискуссией, которая должна быть направлена на выявление дефектов, а не на решение проблем, а также за то, чтобы дискуссия не ушла в сторону в дебри стилистических особенностей и опечаток. Координатор также работает вместе с автором над предложенными изменениями, чтобы все дефекты и проблемы, поднятые в ходе экспертизы, были рассмотрены должным образом.

Читатель Одному из проверяющих назначается роль читателя. На совещании он пересказывает положения спецификации и элементы модели — по одному за раз. Затем другие участники указывают на возможные дефекты и проблемы. Выражая требование своими словами, читатель дает трактовку требования, которая может отличаться от интерпретации других членов команды. Это хороший способ выявления неясностей, возможных недостатков или предположений. Важно подчеркнуть ценность того, что роль читателя поручается не автору. В менее формальных типах рецензирования роль читателя отсутствует, а его роль выполняет координатор, которые последовательно проходит продукт и предлагает участникам делать свои комментарии — по одному разделу за раз.

Секретарь Секретарь, или протоколист, использует стандартные формы для документирования возникающих вопросов и дефектов, выявленных в ходе совещания. Секретарь должен вслух прочитать или показать свои записи (путем отображения на экране или представление на общий обзор в веб-конференции), чтобы убедиться в их точности. Другие участники экспертизы должны помочь протоколисту зафиксировать суть каждой проблемы так, чтобы автор смог четко уяснить природу и местоположение каждой проблемы для ее последующего эффективного и правильного решения.

Входные критерии

Вы готовы к экспертизе документа о требованиях в том случае, если он удовлетворяет определенным предварительным условиям. Набор *входных критериев* (entry criteria) четко обрисовывает авторам то, как следует подготовиться к экспертизе. Они также удерживают команду от траты времени на проблемы, которые следует решать до начала экспертизы. Координатор использует входные критерии в качестве контрольного списка до того, как принять решение о начале экспертизы. Вот несколько входных критериев для составления документации по требованиям:

- документ должен соответствовать стандартному шаблону и не иметь явных проблем с орфографией, грамматикой и форматированием;
- номера строк или другие уникальные идентификаторы следует напечатать в документе, чтобы облегчить ссылки на определенные элементы в течение совещания;

- все нерешенные вопросы помечаются значком *«TBD»* (to be determined — необходимо определить) или доступны в средстве отслеживания дефектов;
- координатор выявляет не более трех существенных дефектов после 10-минутной проверки выбранного фрагмента документа.

Этапы экспертизы

Экспертиза — это многоэтапный процесс, как показано на рис. 17-2. Требования можно проверять небольшими порциями за раз, например назначенными конкретным итерациям разработки, постепенно охватывая все требования. Цель каждого этапа кратко описана далее в этом разделе.

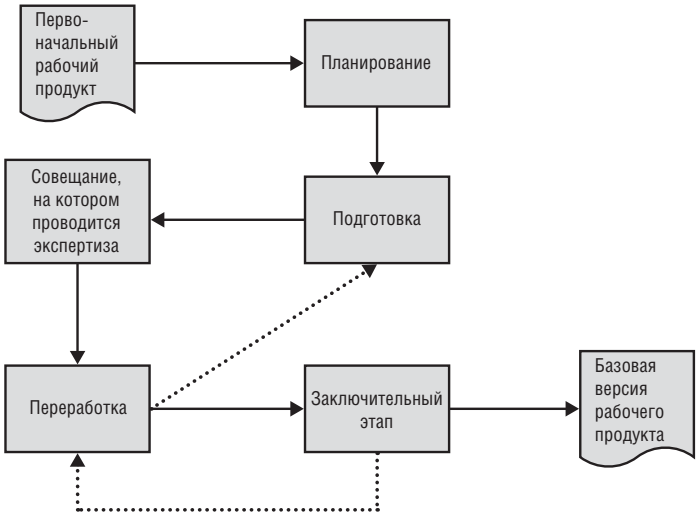


Рис. 17-2. Экспертиза — многоэтапный процесс. Пунктирные линии указывают на то, что определенные этапы экспертизы могут быть выполнены несколько раз

Планирование Экспертизу совместно планируют автор и координатор. Они определяют состав участников, материалы, которые проверяющие должны получить до начала первого совещания, количество времени, необходимого для охвата всего материала, а также, когда нужно планировать совещания. Количество обнаруженных дефектов очень зависит от темпов работы (Gilb и Graham, 1993). Как видно на рис. 17-3, чем медленнее изучается спецификация требований к ПО, тем больше дефектов удается выявить. (Весьма распространено альтернативное толкование этой связки: «Темпы проверки снижаются, если вы обнаруживаете много ошибок». До сих пор так и не решили, что здесь причина, а что следствие.) Поскольку ни одна команда не может тратить бесконечное количество времени на проверку требований, выберите подходящий темп, принимая во внимание риск пропуска важных дефектов. Чаще всего рекомендуется читать от двух до четырех страниц в час, хотя оптимальная скорость, при которой достигается максимальный эффект ниже примерно в два раза (Gilb и Graham, 1993). При выборе темпа работы учитывайте следующие факторы:

- данные предыдущей экспертизы, показывающие эффективность экспертизы в зависимости от скорости;
- объем текста на каждой странице;
- сложность требований;
- риск и последствия того, что ошибка останется незамеченной;
- насколько критично для успеха проекта проверить весь материал;
- квалификация автора спецификации требований к ПО.

Подготовка До совещания автор должен поделиться с экспертами общей информацией, чтобы они понимали контекст проверяемых элементов, а также цели автора в этой экспертизе. После этого каждый из экспертов исследует продукт, чтобы определить возможные дефекты и возникающие вопросы; для этого они используют контрольные списки типичных дефектов (они описаны далее в этой главе) и другие приемы анализа (Wiegers, 2002). До 75% дефектов, обнаруживаемых при экспертизе, выявляются на этапе подготовки (Humphrey, 1989), поэтому этот этап пропускать нельзя. При подготовке могут пригодиться приемы, описанные в разделе «Поиск упущенных требований» главы 7. На индивидуальную подготовку нужно выделить еще примерно половину того времени, что запланировано на совещания.

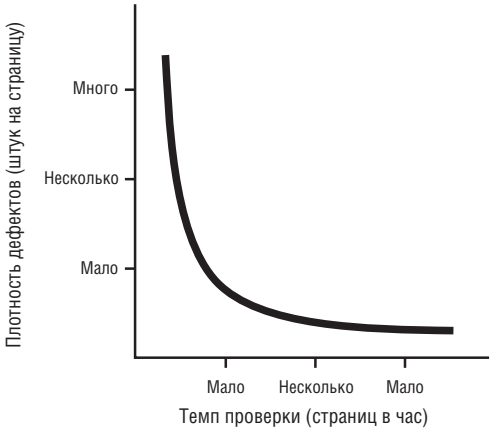


Рис. 17-3. Количество обнаруженных дефектов зависит от темпа инспектирования

> **Внимание!** Не следует начинать совещание, если участники еще не изучили продукт самостоятельно. Неэффективные совещания могут породить ошибочное мнение, что экспертиза не что иное, как пустая трата времени.

Совещание На совещании читатель знакомит членов команды с каждым требованием спецификации — по одному за раз, — перефразируя их своими словами. По мере выявления дефектов и других проблем, секретарь заносит их в форму, которая для автора требований становится руководством к действию. Цель совещания — выявить в документе как можно больше ошибок. Совещание не должно длиться более двух часов, поскольку работу уставших людей нельзя назвать продуктивной. Если же вам нужно больше времени, следует запланировать еще одно совещание.

После изучения всего материала команда принимает решение: следует ли принять документ без изменений, с незначительными поправками или указать на необходимость значительной переработки. Решение «необходимы значительные изменения» свидетельствует о наличии серьезных проблем в процессе разработки требований или о том, что бизнес-аналитику, написавшему требования, нужно еще подучиться. Возможно, стоит изучить уже проделанную работу и попытаться найти выход до того, как приступать к следующему этапу работы над спецификацией (Kerth, 2001). Если нужна серьезная правка, в команде могут решить повторно изучить части продукта, требующие значительной переделки, как показано пунктирной линией между блоками «Переработка» и «Подготовка» на рис. 17-2.

Иногда проверяющие сообщают только о поверхностных и незначительных проблемах. Вдобавок проверяющие легко отвлекаются на дискуссии о том, действительно ли обсуждаемый вопрос является дефектом, на споры о границах проекта и на обсуждение возможных решений проблем. Это все, конечно, полезно, однако эти действия отвлекают проверяющих от непосредственной цели совещания — выявления серьезных дефектов и возможностей улучшения.

Переработка Практически на каждом этапе работы, связанным с контролем качества, выявляются определенные дефекты. После совещания автору следует отвести определенное время на переработку требований. Исправление в будущем дефектов требований обойдется дороже, поэтому именно сейчас следует разрешить неясности, устранить двусмысленности и заложить основу успешного проекта по разработке.

Заключительный этап На этом заключительном этапе экспертизы координатор или специально назначенный член команды работает с автором, чтобы убедиться, что все поднятые проблемы разрешены и все ошибки исправлены соответствующим образом. Заключительный этап завершает процесс экспертизы, на этом этапе координатор определяет, выполняются ли выходные критерии экспертизы. На этом этапе может обнаружиться, что некоторые изменения были сделаны не полностью или некорректно, что требует дополнительных переделок, как показано пунктирной линией между блоками «Заключительный этап» и «Переработка» на рис. 17-2.

Выходные критерии

В ходе экспертизы определяются выходные критерии, которые должны быть выполнены, прежде чем координатор объявит о том, что экспертиза (а не просто совещание) закончена. Вот перечень стандартных выходных критериев экспертизы:

- все возникшие вопросы были сняты;
- все изменения, внесенные в документ и соответствующие продукты, были внесены корректно;
- все открытые проблемы разрешены или для каждой открытой проблемы указаны процесс разрешения, дата и исполнитель.

Контрольные списки дефектов

Чтобы помочь экспертам обнаружить типичные ошибки в проверяемых продуктах, разработайте контрольный список дефектов для каждого типа документа о требованиях, создаваемого в вашей компании. С помощью таких контрольных списков вы привлечете внимание проверяющих к часто возникающим проблемам с требованиями. Контрольные списки служат напоминанием. Со временем люди усваивают определенный порядок и уже по привычке ищут соответствующие проблемы в каждом сеансе проверки.

Полнота

- ❑ Включены ли в требования все известные потребности клиента или системы?
- ❑ Отсутствует ли в требовании какая-либо необходимая информация? Если да, помечена ли она значком «TBD»?
- ❑ Определены ли все алгоритмы, соответствующие функциональным требованиям?
- ❑ Определены ли все внешнее оборудование, ПО и интерфейсы взаимодействия?
- ❑ Задокументировано ли ожидаемое поведение системы для всех предполагаемых ошибочных условий?
- ❑ Обеспечивают ли требования адекватную основу для дизайна и тестирования?
- ❑ Указан ли приоритет реализации каждого требования?
- ❑ Находится ли каждое требование в рамках проекта, выпуска или итерации?

Корректность

- ❑ Конфликтуют ли какие-либо требования с другими требованиями или повторяют их?
- ❑ Написано ли каждое требования ясным, точным и недвусмысленным языком?
- ❑ Можно ли проверить каждое требование с помощью тестирования, демонстрации, просмотра или анализа?
- ❑ Все ли перечисленные сообщения об ошибках уникальны и выразительны?
- ❑ Являются ли все требования действительно требованиями, а не решениями или ограничениями?
- ❑ Можно ли реализовать все требования, не выходя за рамки установленных ограничений?

Атрибуты качества

- ❑ Все ли цели по производительности, безопасности и защите сформулированы надлежащим образом?
- ❑ Все ли остальные атрибуты качества задокументированы и оценены количественно с указанием приемлемых компромиссов?
- ❑ Все ли критичные по времени функции определены и задан график реализации для них?
- ❑ Решены ли надлежащим образом вопросы интернационализации и локализации?
- ❑ Все ли требования к качеству поддаются измерению?

Организация и возможность отслеживания

- ❑ Все ли требования организованы логичным и доступным образом?
- ❑ Корректны ли все перекрестные ссылки на другие требования и документы?
- ❑ Написаны ли все ли требования на одном и том же надлежащем уровне детализации?
- ❑ Каждое ли требование идентифицировано уникально и корректно?
- ❑ Указана ли для каждого функционального требования ссылка на источник (например, на системное требование или бизнес-правило)?

Другие проблемы

- ❑ Не пропущены ли какие-то варианты использования или потоки процессов?
- ❑ Не пропущены ли в вариантах использования какие-либо альтернативные направления, исключения или другая информация?
- ❑ Определены ли все бизнес-правила?
- ❑ Не упущены ли какие-либо наглядные модели, которые необходимы для ясности и полноты?
- ❑ Не пропущены ли какие-либо спецификации отчетов и полны ли они?

Рис. 17-4. Контрольные списки дефектов для спецификации требований к ПО

На рис. 17-4 показан контрольный список для проверки спецификации требований к ПО (его можно взять в прилагаемом к этой книге материале). При создании конкретных представлений или моделей требований можно расширить и уточнить указанные здесь элементы контрольного списка.

Бизнес-требования, например в форме документа о видении и границах проекта, могут требовать собственного контрольного списка. Сесилия Хоффман и Ребекка Бергес (Ce-cilie Hoffman и Rebecca Burgess, 2009) предоставляют несколько подробных контрольных списков, в том числе для проверки требований к ПО по отношению к бизнес-требованиям.

Весь длинный контрольный список запомнить невозможно. Если в списке более шести или восьми элементов, проверяющему скорее всего потребуется несколько раз пройти материал, что просмотреть все, указанное в списке; большинство проверяющих не будет себя утруждать этим. Сократите списки, сообразуясь с нуждами вашей компании, и измените их в соответствии с тем, какие проблемы в ваших требованиях повторяются чаще всего. Проведенные исследования показали, что такое распределение обязанностей — естественно, когда экспертов снабжают четкими указаниями или сценариями, которые помогают им при поиске ошибок — более эффективно, чем когда эксперты работают по одинаковым контрольным спискам. (Porter, Votta и Basili, 1995).

Советы по рецензированию требований

В главе 8 книги Карла Вигерса «More About Software Requirements: Thorny Issues and Practical Advice» («Подробнее о требованиях к ПО: сложные вопросы и практические советы», Microsoft Press, 2006) содержатся советы по совершенствованию процессов рецензирования требований. Следующие рекомендации применимы как к формальному, так и неформальному рецензированию в ваших проектах, и независимо от того, где хранятся требования — в традиционных документах, в средстве управления требованиями или в любой другой осязаемой форме.

Правильно планируйте проверку Когда кто-то просит прорецензировать документ, хочется начать с самого начала и последовательно читать до самого конца. Но это делать не потребуется. Потребители спецификации требований не будут читать их от корки до корки, как книгу, впрочем как и рецензенты. Пригласите разных рецензентов поработать над разными разделами документов.

Начинайте как можно раньше Начинайте отдавать наборы требования на рецензирование, когда готово, скажем, 10% работы, а не когда все требования «готовы». Обнаружение серьезных дефектов как можно раньше и выявление систематических проблем в способе написания требований — мощный метод предотвращения, а не только поиска, дефектов.

Выделите достаточно времени Дайте проверяющим достаточно времени на рецензирование — как в смысле реальных часов, так и календарного времени. У них есть другая важная работа, а рецензирование им приходится втискивать между другими задачами.

Предоставьте контекст Сообщите рецензентам контекст документа и, возможно, проекта, если они не работают над тем же проектом. Найдите рецензентов, которые могут предоставить полезный вклад благодаря своим зна-

ниями. Например, у вашего знакомого из другого проекта может быть цепкий глаз на серьезные пробелы в требованиях, даже если он не очень близко знаком с проектом.

Определите границы рецензирования Сообщите рецензентам, какой материал изучать, на что обращать особое внимание и какие дефекты искать. Предложите им использовать контрольный список дефектов, похожий на тот, что описан в предыдущем разделе. Можно повысить доступность и квалификацию команды рецензентов, попросив разных проверяющих проверить разные разделы или использовать различные части контрольных списков.

Ограничьте повторное рецензирование Не просите никого просматривать один и тот же материал более трех раз. Рецензент может утомиться от просмотра одного и того же и после третьего цикла не заметить серьезные проблемы из-за «замыленности глаза». Если вам нужно, чтобы рецензент просмотрел какую-то часть несколько раз, каждый раз выделяйте изменения, чтобы он мог сосредоточиться именно на них.

Определяйте приоритеты рецензируемых областей Определите приоритеты тех частей требований, которые отличаются высоким риском или относятся к редко используемой функциональности. Также поищите области требований, где найдено немного дефектов. Это может означать, что эти части еще не прошли рецензирование, а не отсутствие в них проблем.

Сложности рецензирования требований

Экспертные оценки можно отнести как к техническим операциями, так и к работе с людьми. Просьба коллег высказать их мнение о том, что не так с вашей работой, — сознательное, а не интуитивное действие. Для введения практики экспертных оценок в компанию, специализирующуюся на разработке ПО, необходимо время. Ниже описаны типичные проблемы, с которыми специалисты компании сталкиваются при рецензировании документации по требованиям, и предложения по их решению (Wiegers, 1998; Wiegers, 2002).

Большой объем документации Перспективу тщательной проверки спецификации требований к ПО, состоящего из нескольких сотен страниц, можно смело назвать обескураживающей. Вам покажется заманчивым пропустить проверку и сразу приступить к конструированию — тоже не самый мудрый выбор. Даже в спецификации среднего размера, как правило, все эксперты тщательно проверят первую часть, несколько наиболее стойких изучат середину, но маловероятно, чтобы кто-то дошел до конца.

Чтобы не перегружать команду, выполняйте рецензирование документ по мере его разработки, не ждите, когда он будет закончен. Определите области повышенного риска, которые требуют тщательной экспертизы, а для менее важных фрагментов хватит и неофициальной рецензии. Попросите определенных экспертов начать с различных частей документа — это позволит окинуть все страницы свежим взглядом. Чтобы оценить, требуется ли проводить экспертизу всей спецификации, исследуйте репрезентативную выборку

фрагментов (Gilb и Graham, 1993). Изучив количество и типы обнаруженных ошибок, вы определите, стоит ли проводить полную проверку.

Большая команда экспертов Многие участники проекта и клиенты заинтересованы в требованиях, поэтому список желающих может оказаться весьма длинным. Однако чем больше команда экспертов, тем труднее составить график совещаний, тем больше на них посторонних разговоров и тем труднее прийти к согласию по каждому вопросу. Однажды я вместе с 13 экспертами был приглашен на совещание, организованное человеком, который явно не понимал важность сохранения небольшой численности команды. Четырнадцати человекам трудно договориться, как выйти из охваченной огнем комнаты, не говоря уже о принятии решения относительно правильности того или иного требования. Попробуйте придерживаться следующих советов при работе с большими командами экспертов:

- убедитесь, что все участники понимают, что их задача — выявлять дефекты, а не повышать квалификацию или отстаивать позицию;
- четко представьте себе, чью точку зрения (клиента, разработчика или тестировщика) представляет каждый участник. Если несколько человек относятся к одной группе, они могут объединить свою информацию и делегировать для участия в совещании кого-то одного;
- соберите несколько небольших групп для параллельной проверки спецификации требований к ПО и объедините их контрольные списки дефектов в один, удалив все повторы. Исследования показали, что экспертиза документа о требованиях, выполняемая несколькими командами, выявляет больше ошибок, чем одна большая команда (Martin и Tsai, 1990; Schneider, Martin и Tsai, 1992; Kosman, 1997). Как правило, результаты параллельных проверок скорее дополняют, чем повторяют друг друга.

Географическое распределение экспертов Все больше компаний разрабатывают продукты, привлекая команды, разъединенные географически. В этом случае проверки проводить труднее. Видеоконференции могут стать эффективным решением, но в телеконференциях вы не можете воспринимать язык тела и мимику других участников, как при личных встречах. Средства вебконференций позволяют обеспечить, чтобы во время обсуждения все рецензенты видели один и тот же материал.

Рецензирование электронного документа, размещенного в общей сетевой папке, — альтернатива традиционным совещаниям экспертов. В этом случае рецензенты используют функции текстового редактора, чтобы ввести свои комментарии в проверяемый документ. (Именно так авторы этой книги Карл и Джой проверяли работу друг друга в процессе написания этой книги.) Каждый комментарий помечается инициалами эксперта, таким образом, каждый любой может ознакомиться с мнением других рецензентов. Полезными также могут быть веб-средства совместной работы. Некоторые средства управления требованиями содержат компоненты поддержки распределенного асинхронного рецензирования, не предусматривающие инте-

рактивных совещаний, Если вы решите не проводить совещания, то имейте в виду, что при этом результативность рецензирования снизится, но это все равно лучше, чем вообще не выполнять рецензирование.

Неготовые рецензенты Одно из условий формальной встречи для рецензирования заключается в том, что все участники должны заранее изучить материал и самостоятельно сформулировать начальный набор проблем. Без такой подготовки вы рискуете, что эксперты будут выполнять весь анализ на встрече «с колес» и скорее всего упустят многие важные проблемы.

В одном проекте 50-страничную спецификацию требований к ПО должны были рецензировать 15 человек — намного больше, чем нужно для эффективной работы. Была выделена неделя, чтобы каждый мог самостоятельно изучить документ и отправить список проблем автору. Неудивительно, что многие вообще не посмотрели спецификацию. Итак, ведущий бизнес-аналитик назначил обязательное совещание, на котором все рецензенты должны были вместе проверить документ. Он вывел спецификацию требований к ПО на экран, приглушил освещение и начал читать требования — одно за другим. Несколько часов спустя участники начали зевать, а их внимание — улетучиваться. Неудивительно, что скорость выявления дефектов упала. Все с нетерпением ждали окончания совещания. Бизнес-аналитик решил отпустить участников и предложил им самостоятельно изучить документ, чтобы следующее совещание прошло более эффективно. Ясно, что перспектива еще раз поучаствовать в мучительном совещании заставила их лучше подготовиться к следующему разу. Подробнее о том, как вовлекать участников в процесс, см. раздел «Семинары» в главе 7.

Прототипы требований

Трудно представить себе, как система будет функционировать при определенных условиях только на основании прочитанной спецификации требований к ПО. Прототипы — ценный инструмент проверки требований. Они позволяют пользователю увидеть, как будут выглядеть некоторые стороны системы, созданной на основе определенных требований. Подробнее о разных типах прототипов и как их применяют для улучшения требований см. главу 15. Здесь мы расскажем, как прототипы могут помочь заинтересованным лицам оценить, будет ли продукт, созданный в соответствии с имеющимися требованиями, удовлетворять их нужды, а также являются ли требования полными, реализуемыми и четко сформулированными..

Все типы прототипов позволяют вам обнаружить отсутствующие требования до начала таких более дорогостоящих мероприятий, как разработка и тестирование. Для прохождения вариантов использования, процессов или функций с целью обнаружения пропущенных или ошибочных требований можно применить просто бумажную модель. Прототипы также помогают проверить, одинаково ли понимают требования все заинтересованные лица. Бывает так, что вы создадите прототип на основе своего понимания требова-

ния только для того, чтобы обнаружить, что из-за неточной формулировки требования рецензенты прототипа не согласятся с вашей трактовкой.

Экспериментальный образец может служить для демонстрации реализуемости требования. Эволюционные прототипы позволяют пользователям увидеть, как будут работать требования после реализации и убедиться, что результат именно тот, что ожидался. Более сложные прототипы, например имитационные модели, позволяют более точно проверять требования, но на их создание требуется больше времени.

Тестирование требований

Тесты, созданные на основе функциональных или пользовательских требований, помогают участникам проекта получить представление об ожидаемом поведении системы. Просто разработка тестов поможет выявить множество проблем с требованиями, даже если вы не станете выполнять тесты на работающем ПО. Будут появляться неясные и двусмысленные требования, потому что вы не сможете описать ожидаемую реакцию систему. Когда аналитики и клиенты вместе создают тесты, они вырабатывают общее понимание того, как продукт будет работать, и приобретают уверенность в том, что требование сформулировано правильно. Тестирование — мощный инструмент проверки и утверждения требований.

> **Внимание!** Остерегайтесь тестировщиков, которые утверждают, что не могут приступить к работе, пока требования еще не готовы, и тех, которые утверждают, что для тестирования ПО им не нужны требования. Тестирование и требования связаны между собой синергетическими отношениями, поскольку они представляют взаимодополняющие взгляды на систему.

Осчастливить рядового Чарли

Как-то раз я попросил Чарли, профи в написании сценариев для UNIX в моей группе, сконструировать простое расширение интерфейса электронной почты для серийной системы отслеживания дефектов, которой мы пользовались. Я написал десяток функциональных требований, описывающих, как должен работать интерфейс электронной почты. Чарли был в восторге. Он написал множество сценариев, но до сих пор не видел ни одного требования в письменной форме.

К сожалению, я промедлил пару недель, прежде чем приступил к написанию тестов для этой функции электронной почты. Разумеется, я допустил ошибку в одном из требований. Я обнаружил ее, поскольку мое представление о том, как функция должна работать, было представлено в 20 тестах, оказалось несовместимо с одним из требований. Раздосадованный, я исправил требование с ошибкой, еще до того, как Чарли завершил разработку, и в законченном варианте не было ни одной ошибки. Обнаружение ошибки до реализации — это маленькая победа, но даже маленькие победы вносят свой вклад в общий успех.

Вы можете заняться созданием концептуальных тестов, основываясь на пользовательских требованиях, уже на ранней стадии процесса разработки (Collard, 1999; Armour и Miller, 2001). Также тесты можно применять для оценки функциональных требований, моделей анализа и прототипов. Тесты должны охватывать нормальные направления всех вариантов использования, альтернативные направления, а также исключения, определенные в ходе выявления требований и анализа. Аналогично, если вы определили потоки бизнес-процессов, тесты должны охватывать этапы бизнес-процессов и все возможные пути принятия решений.

Эти концептуальные (или абстрактные) варианты тестирования не зависят от реализации. Для примера рассмотрим вариант использования «Просмотр заказа» в Chemical Tracking System. Концептуальные тесты могут быть следующими:

- пользователь вводит номер заказа, который хочет просмотреть, этот заказ существует, пользователь разместил заказ. Ожидаемый результат: отображение подробностей заказа;
- пользователь вводит номер заказа, который хочет просмотреть, этот заказ не существует. Ожидаемый результат: появляется сообщение: «К сожалению, заказ не найден»;
- пользователь вводит номер заказа, который он хочет просмотреть, этот заказ существует, но он размещен не этим пользователем. Ожидаемый результат: появляется сообщение: «К сожалению, это не ваш заказ».

В идеале разработчик пишет функциональные требования, а тестировщик — тесты на основе одних и тех же материалов: пользовательских требований, показанных на рис. 17-5. Неясности в пользовательских требованиях и различные интерпретации выливаются в несогласованность представлений на основе функциональных требований, моделей и вариантов тестирования. По мере того, как разработчики преобразовывают требования в пользовательский интерфейс и технический дизайн, тестировщики могут разработать концептуальные тесты в детально проработанные процедуры тестирования для официального тестирования системы (Hsia, Kung и Sell, 1997).

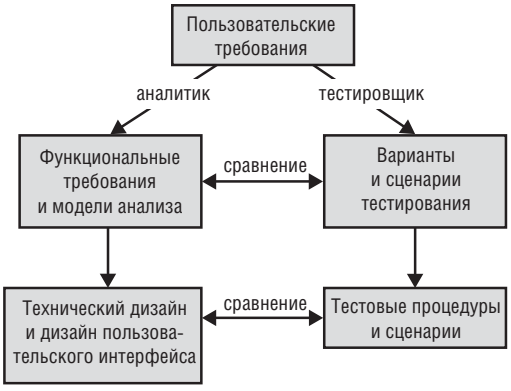


Рис. 17-5. Разработка и тестирование продуктов имеют один общий источник

Давайте посмотрим, как команда разработчиков Chemical Tracking System связали вместе требования и визуальные модели с первой стадией создания тестов. Далее рассматриваются несколько видов относящейся к требованиям информации, которые связаны с задачей заказа химиката.

Бизнес-требование Как говорится в главе 5, одна из основных бизнес-целей Chemical Tracking System такова:

Сократить затраты на закупку химикатов на 25% к концу первого года.

Вариант использования продукта Вариант использования продукта, поддерживающий это бизнес-требование, называется «Заказ химиката». Он позволяет пользователю заказать контейнер с химикатом, который уже имеется в наличии на складе химикатов. Вот как звучит описание варианта использования на рис. 8-3 в главе 8:

Сотрудник, разместивший заказ на химикат, указывает в заказе необходимый химикат, вводя его название или идентификатор или импортируя его структуру из соответствующего графического средства. Система выполняет заказ, предлагая контейнер с химикатом со склада или позволяя создать заказ у поставщика.

Функциональное требование Здесь показана некоторая функциональность, связанная с этим вариантом использования.

1. *Если на складе есть контейнеры с химикатами, на которые поступил заказ, система отобразит список доступных контейнеров.*
2. *Пользователь либо выберет один из перечисленных контейнеров, либо попросит разместить заказ нового контейнера у продавца.*

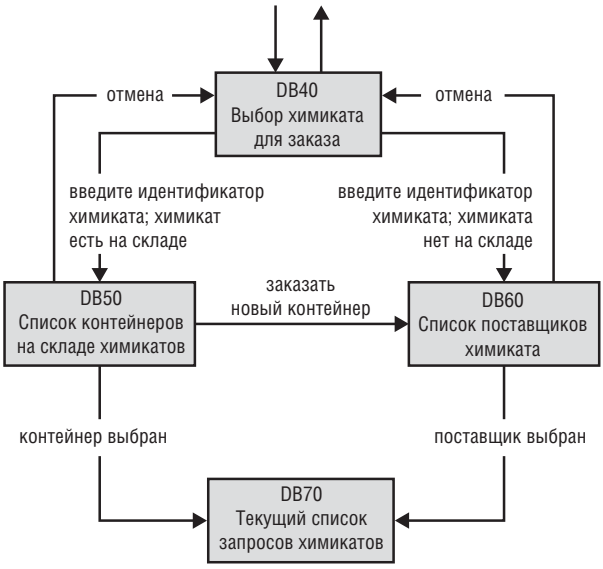


Рис. 17-6. Фрагмент карты диалоговых окон для варианта использования «Заказ химиката»

Карта диалоговых окон На рис. 17-6 показана часть карты диалоговых окон для варианта использования «Заказ химиката», которые имеют отно-

шение к этой функции. Как уже обсуждалось в главе 12, прямоугольники на этой карте диалоговых окон представляют изображения пользовательского интерфейса, а стрелки показывают возможные пути перехода от одного изображения к другому. Эта карта диалоговых окон была создана на довольно поздних этапах разработки требований, когда участники проекта начали определять конкретные экраны, меню, диалоговые окна и другие элементы карты, чтобы можно было дать им конкретные названия и начать думать о возможной архитектуре пользовательского интерфейса.

Тест Поскольку для этого варианта использования существует несколько возможных путей выполнения, можно придумать несколько тестов для обработки нормального направления, альтернативных направлений и исключений. Ниже показан один тест, когда пользователю представляется список контейнеров, доступных на складе.

В диалоговом окне DB40 введите действительный идентификатор химиката; на складе имеются два контейнера с этим химикатом. Откроется диалоговое окно DB50, в котором показаны эти два контейнера. Выберите второй контейнер. DB50 закроется, и контейнер 2 будет добавлен в конец списка текущих запросов на химикаты в диалоговом окне DB70.

Рамеш, руководитель тестирования Chemical Tracking System, написал несколько аналогичных тестов, исходя из своего понимания варианта использования. Такие абстрактные тесты не зависят от деталей реализации. В них не разъясняется ввод данных в поля, щелчки кнопок или другие специфичные детали взаимодействия. По мере разработки пользовательского интерфейса, Рамеш возвращался к этим абстрактным вариантам тестирования, создавая более точные процедуры тестирования.

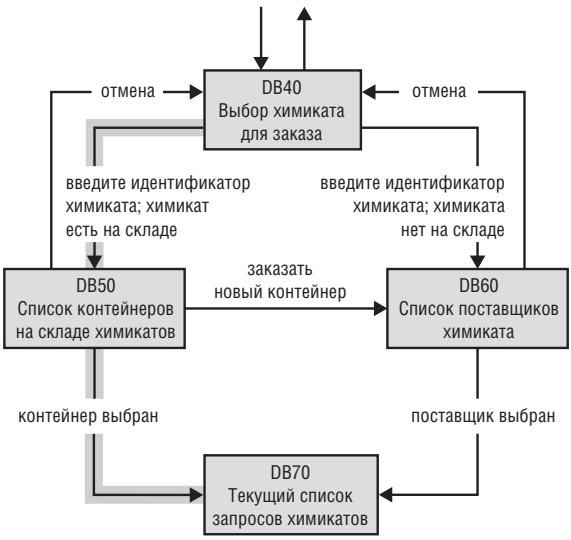


Рис. 17-7. Отслеживание теста на карте диалоговых окон для варианта использования «Заказ химиката»

А теперь самое интересное — тестирование требований. Сначала Рамеш установил соответствие между каждым вариантом тестирования и функциональными требованиями. Он проверил и убедился, что при существующем наборе требований все тесты «выполняются». Он также убедился, что каждому функциональному требованию соответствует по крайней мере один вариант тестирования. Затем Рамеш проследил и отметил с помощью маркера путь исполнения каждого теста по карте диалоговых окон. Выделенная линия на рис. 17-7 показывает, путь предыдущего варианта тестирования на карте диалоговых окон.

Отслеживая путь исполнения для каждого теста, можно найти некорректные или пропущенные требования, исправить ошибки на карте диалоговых окон и отшлифовать тесты. Предположим, что после такого «исполнения» всех тестов линия на карте, помеченная как «заказать новый контейнер», от DB50 к DB60 на рис. 17-6 не была выделена. Возможны два объяснения этому:

- перемещение от DB50 к DB60 не допускается поведением системы. Бизнес-аналитик должен убрать эту линию с карты диалогов. Если в спецификации есть требование, в котором указан этот переход, то это требование также необходимо удалить;
- перемещение допустимо поведением системы, но вариант тестирования, который демонстрирует это поведение, отсутствует.

Точно так же, предположим, что тест основывается на собственной интерпретации варианта использования тестировщиком и позволяет перейти непосредственно от диалогового окна DB40 к DB70. Однако в диалоговом окне на рис. 17-6 такой линии нет, и поэтому при существующих требованиях тест «выполнить» нельзя. И опять же, возможны две интерпретации, и вам необходимо определить, какая из них верна:

- переход от DB40 к DB70 не допускается поведением системы, следовательно, тест неправильный;
- переход от DB40 к DB70 допустимо поведением системы, но на карте диалоговых окон и, возможно, в спецификации требований к ПО отсутствует требование, позволяющее выполнить тест.

В этих примерах аналитик и тестировщик объединили требования, модели анализа и тесты для выявления пропущенных, ошибочных или ненужных требований задолго до написания какого-либо кода. Концептуальное тестирование требований к ПО — мощный прием управления затратами и графиком проекта, так как он позволяет выявлять неясности и ошибки в требованиях как можно раньше. Как сказал Росс Коллард (Ross Collard, 1999):

Есть два варианта отличного взаимодействия вариантов использования и тестов: если варианты использования написаны полно, аккуратно и ясно, то процесс составления тестов становится крайне простым, а если варианты использования не в лучшей форме, то попытка составления тестов поможет вам отладить вариант использования продукта.

Утверждение требований с применением критериев приемки

Разработчики ПО могут быть уверены, что они создали идеальный продукт, однако последнее слово за клиентом. Клиенты должны определить, удовлетворяет ли система *критерию приемки* (acceptance criteria). Критерий приемки — и, следовательно, приемочное тестирование — является показателем, удовлетворяет ли продукт задокументированным требованиям и годится ли он для использования в предполагаемой операционной среде (Hsia, Kung и Sell, 1997; Leffingwell, 2011; Pugh, 2011). Делегирование разработки тестов на приемлемость пользователям — эффективная стратегия разработки требований. Чем раньше в процессе разработки пользователи продумают тесты на проверку приемлемости, тем скорее удастся отловить дефекты в требованиях и разрабатываемом ПО.

Критерии приемки

Работа с клиентами над разработкой критериев приемки позволяет утвердить как требования, так и само решение. Если клиент не может описать, как он оценит, что конкретное требование удовлетворено системой, значит, требование сформулировано недостаточно ясно. Критерии приемки определяют минимальные условия, при которых приложение можно считать готовым к промышленной эксплуатации.

Мышление в терминах критериев приемки знаменует изменение формулировки вопроса с «Что вам нужно делать с помощью системы?» к «Как вы делаете вывод о том, что система удовлетворяет вашим потребностям?». Поощряйте пользователей при определении критериев приемки использовать мнемоническое правило SMART, то есть они должны быть конкретными (Specific), измеримыми (Measurable), достижимыми (Attainable), актуальными (Relevant) и ограниченными во времени (Time-sensitive). Критерии надо определять так, чтобы разные объективные наблюдатели приходили к одному мнению относительно того, выполняются ли эти критерии. Критерии приемки ориентированы на бизнес-цели заинтересованных лиц и условия, которые позволяют куратору проекта объявить победу. Это важнее, чем простая поставка продукта, соответствующего спецификации требований, но не обязательно решающего бизнес-задачи заинтересованного лица.

Определение критериев приемки это больше, чем утверждение, что все требования реализованы или что все тесты пройдены. Приемочные тесты составляют львиную часть критериев приемки. Критерии приемки могут также охватывать такие аспекты, как:

- Определенная высокоприоритетная функциональность, которая должна быть в наличии и работать надлежащим образом и без которой продукт не может приниматься и использоваться. (Другая запланированная функциональность может поставляться позже или не совсем хорошо работающие

функции можно исправить без задержки начального выпуска.)

- Обязательные нефункциональные критерии или метрики качества, которые должны удовлетворяться. (Некоторые атрибуты качества должны быть удовлетворены хотя бы минимальным образом, а улучшения удобства использования, внешнего вида и производительности можно отложить. От продукта может требоваться удовлетворение метрик качества, таких как определенный минимальный период работы без отказа.)
- Остающиеся открытые проблемы и дефекты. (Можно оговорить, что в реализации открытых высокоприоритетных требований не должно быть дефектов выше определенного уровня важности, хотя менее важные дефекты могут быть.)
- Определенные юридические, нормативные или контрактные условия. (Только при их полном выполнении продукт может считаться приемлемым.)
- Требования поддержки переноса и инфраструктуры или другие проектные (не относящиеся к продукту) требования. (Например, до выпуска решения может требоваться обеспечить наличие обучающих материалов и завершение преобразования данных.)

Также бывает полезным подумать об условиях «критериев отказа» или результатах оценки, которые могут заставить заинтересованное лицо не признать систему готовой к поставке. Следует опасаться конфликтующих критериев приемки, например если выполнение одного критерия не позволяет удовлетворить другой. На самом деле, выявление противоречивых критериев приемки — один из способов выявления противоречивых требований.

В проектах гибкой разработки критерии приемки основываются на пользовательских историях. Как говорит Дин Леффингуэлл (Dean Leffingwell, 2011):

Критерии приемки это не функциональные или модульные тесты, — это условия удовлетворительности системы. Функциональные и модульные тесты предусматривают намного более глубокую проверку всех функциональных направлений, исключительных направлений, граничных условий и связанной функциональности, относящейся к истории.

В принципе, если все критерии приемки пользовательской истории выполняются, владелец продукта принимает решении о том, что пользовательская история реализована. Поэтому клиенты должны очень точно формулировать важные для них критерии приемки.

Приемочные тесты

Приемочные тесты составляют львиную часть критериев приемки. Ключевым пользователям при принятии решения о способе оценки приемлемости системы стоит принять во внимание наиболее общие и важные варианты использования, когда они решают, каким образом оценивать приемлемость ПО. Приемочные тесты фокусируют на нормальных направлениях вариантов использования и их исключениях, а не на более редких альтернативных

направлениях. Кен Пью (Ken Pugh, 2011) предлагает массу рекомендаций по написании приемочных тестов, основанных на требованиях.

В проектах гибкой разработки часто создают приемочные тесты вместо подробных функциональных требований. Каждый тест описывает, как пользовательская история должна работать будучи реализованной в виде ПО. Так как приемочные тесты по большей части заменяют собой подробные требования, в проекте гибкой разработки они должны охватывать все сценарии успеха и неудачи (Leffingwell, 2011). Ценность написания приемочных тестов заключается в том, что они заставляют пользователей думать, как система будет себя вести после реализации. Проблема с написанием *только* приемочных тестов в том, что требования существуют только в головах людей. Отсутствие документирования и сравнения альтернативных представлений требований — пользовательских и функциональных требований, моделей анализа и тестов — может привести к тому, что вы пропустите некоторые ошибки, несоответствия и пробелы.

При малейшей возможности старайтесь автоматизировать тестирование приемлемости. Это упростит вам жизнь, когда тесты придется повторять после внесения изменений и добавления дополнительной функциональности в следующих итерациях или выпусках. Приемочные тесты должны ориентироваться на нефункциональные требования. Они должны подтверждать, что цели по производительности достигаются на всех платформах, система соответствует стандартам легкости и простоты использования и все соответствующие пользовательские требования реализованы.

Часть приемочного тестирования может выполняться пользователями вручную. Тесты пользовательского приемочного тестирования должны выполняться после того, когда набор функциональности считается готовым к выпуску. Это позволяет пользователям попробовать и познакомиться с работающим ПО до его официального выпуска. Клиент или сторонник продукта должен выбрать тесты для пользовательского приемочного тестирования, которые охватывают самый рискованные части системы. Приемочные тесты должны подтвердить, что решение делает то, что должно. Не забудьте при создании этих тестов использовать близкие к реальности тестовые данные. Представьте, что тестовые данные, использованные для создания отчета о продажах, не похожи на реальные. Пользователь, выполняющий приемочное тестирование, может ошибочно сообщить о дефекте просто потому, что отчет ему кажется неправильным или он заметил неправильное вычисление из-за некорректности данных.

> **Внимание!** Пользовательское тестирование приемлемости не заменит полного тестирования системы на основании требований, при котором тестируются все нормальные пути и пути исключений, а также большое количество комбинаций данных, граничных условий и других мест, где могут скрываться дефекты.

Недостаточно просто написать требования. Вы должны убедиться, что требования *правильные* и нужны и что они *достаточно хороши*, чтобы

стать основой для дизайна, сборки, тестирования и управления проектом. Планирование тестов приемлемости, неформальное рецензирование требований, экспертиза спецификации требований к ПО и тестирование требований поможет вам собрать продукт высокого качества быстрее и за меньшие деньги, чем когда бы то ни было.

Что дальше?

- Выберите наугад страницу в спецификации требований к ПО вашего проекта. Попросите представителей различных групп заинтересованных в проекте лиц тщательно проверить ее с помощью контрольного списка дефектов на рис. 17-4 на предмет выявления проблем.
- Если в ходе этой пробной проверки обнаружено столько ошибок, что рецензенты забеспокоились о качестве продукта, убедите представителей пользователей и разработчиков проверить всю спецификацию требований к ПО. Чтобы эта проверка была максимально результативной, обучите команду проведению экспертизы.
- Определите концептуальные тесты для варианта использования или для той части спецификации требований к ПО, которая еще не реализована в виде кода. Спросите у представителей пользователей, отражено ли, по их мнению, в вариантах тестирования предполагаемое поведение системы. Убедитесь, что вы определили все функциональные требования, которые будут разрешены конкретным вариантом тестирования, и что в спецификации нет лишних требований.
- Совместно со сторонниками продукта определите критерии, которые они и их коллеги будут использовать для оценки приемлемости системы. Попросите их определить приемочные тесты, которые будут служить для оценки готовности.

Глава 18

Повторное использование требований

Сильвия, менеджер продукта в компании Tailspin Toys, встретилась с руководителем разработки семейства планшетных приложений для музыкантов. «Прасад, я только что узнала, что компания Fabrikam Inc. планирует выпустить более крупную версию своего планшета, который называется Substrate. Сейчас наш эмулятор гитарного усилителя поддерживает работу на их планшете поменьше под названием ScratchPad. Нам нужна версия для Substrate. На большем экране мы можем сделать больше. На Substrate будет устанавливаться новая версия их операционной системы, которая может работать на обоих планшетах».

«Класс! Это круто, — обрадовался Прасад. — Я хотел бы разместить на экране больше элементов управления усилителем. Кроме того, мы можем сделать кнопки и регуляторы больше и удобнее. Мы можем повторно использовать много базовой функциональности из версии эмулятора для ScratchPad. А если Fabrikam не поменяли API-интерфейс операционной системы, мы можем также повторно задействовать часть кода. Наверное, мы избавимся от части функций из версии для ScratchPad, которыми клиенты не пользуются. Мы можем добавить гибридный усилитель для переключения между транзисторным и ламповым звуком из веб-версии, но нам потребуется внести ряд изменений, чтобы вписаться в частотную характеристику планшета. Это будет весело!»

Повторное использование для тех, кому нужна повышенная продуктивность ПО. Люди часто задумываются о повторном использовании кода, но многие другие компоненты проекта ПО поддаются повторному использованию. Повторное применение требований может повысить производительность и улучшить качество, а также обеспечить лучшую совместимость между связанными системами.

Повторное использование означает применение работы, которая была сделана раньше, в том же или предыдущем проекте. Каждый раз, когда удается начать не с нуля, ваш проект получает значительный задел с самого начала. Проще всего повторно использовать требование, скопировав его из существующей спецификации. А самый сложный способ — повторно использовать целый функциональный компонент, от требований, до дизайна, кода и

тестов. Между этими крайними точками существует масса других вариантов повторного использования.

Но оно не обходится даром. У повторного использования собственные риски — как при применении существующих элементов, так и при создании элементов с хорошими возможностями повторного использования. Скорее всего на разработку высококачественных повторно используемых требований потребуется больше времени и усилий, чем на написание требований, рассчитанных только на текущий проект. Одно исследование показало, что несмотря на очевидные преимущества, только в половине опрошенных организаций требования используются повторно, в основном из-за низкого качества уже имеющихся требований (Chernak, 2012). Организация, в которой серьезно настроены на повторное использование, должна установить какую-то инфраструктуру, чтобы существующие знания о высококачественных требованиях были доступны будущим бизнес-аналитикам, а также всячески развивать культуру, в которой ценится повторное использование.

В этой главе описываются несколько видов повторного использования, определены некоторые классы информации требований, которые можно повторно использовать в разных контекстах и предлагаются советы по повторному использованию требований. Также обсуждаются вопросы обеспечения возможности повторного использования требований. В конце главы рассказывается о препятствиях на пути к эффективному повторному использованию, а также условия успешного повторного использования существующего набора требований.

Зачем нужно повторное использование требований?

К преимуществам повторного использования относятся сокращение сроков создания требований, снижение затрат на разработку, единообразие внутри приложения и между разными приложениями, повышение производительности команды, снижение числа дефектов и сокращение объема переделок. Повторное использование надежных требований может сэкономить время рецензирования, сократить цикл одобрения и ускорить другие операции проекта, в том числе тестирование. Повторное использование может обеспечить более качественную оценку усилий на реализацию, если у вас есть информация о реализации таких же требований в предыдущем проекте.

С точки зрения пользователя повторное использование требований может обеспечить функциональное единообразие в рамках семейства продуктов или бизнес-приложений. Представьте себе возможность форматирования блоков текста путем применения одинаковых стилей, интервалов и других свойств набора связанных приложений. Для обеспечение единообразия вида и поведения требуется повторное использование как функциональных требований, так и требований к удобству использования. Такое единообразие

может сократить срок освоения и предотвратить неудовлетворенность пользователя работой программы. Это также экономит время заинтересованных лиц, которым не придется многократно определять похожие требования.

Даже если реализация отличается в различных средах, требования могут оставаться неизменными. На веб-сайте авиакомпании может быть функциональность, позволяющая пассажирам регистрироваться на рейс, оплачивать дополнительные удобства и печатать посадочные талоны. Одновременно у компании могут быть терминалы регистрации в аэропортах. Функциональность в обеих этих точках практически одинакова, а значит, ее можно повторно использовать в двух продуктах, несмотря на различия в реализации и пользовательском интерфейсе.

Виды повторного использования требований

Мы представляем себе несколько типов повторного использования требований. Бывает так, что бизнес-аналитик замечает, что представленное пользователем требование похоже на требование из предыдущего проекта. Скорее всего он сможет взять существующее требование и приспособить его к новому проекту. Такое повторное использование «по месту» часто выполняют опытные бизнес-аналитики, у которых хорошая память и большой запас ранее разработанных требований. В других случаях бизнес-аналитик может использовать во время выявления требований какие-то имеющиеся требования, чтобы помочь пользователям в определении того, что нужно от новой системы. Проще изменить что-то уже существующее, чем создавать все с нуля.

В табл. 18-1 описываются измерения повторного использования требований: объем повторно используемых активов, объем необходимых изменений и применяемый механизм повторного использования. Анализируя возможность повторного использования, подумайте о вариантах в каждом из этих измерений, которые наиболее практичны и уместных для решения имеющихся задач.

Табл. 18-1. Три измерения повторного использования требований

Измерение	Варианты
Объем повторного использования	Формулировка одного требования
	Требование и его атрибуты
	Требование с атрибутами, контекстом и связанной информацией, такой как определения данных, определения терминов, приемочные тесты, предположения, ограничения и бизнес-правила
	Набор связанных требований
	Набор требований и связанные элементы дизайна
	Набор требований и связанные элементы дизайна, код и тесты

Табл. 18-1. (*окончание*)

Измерение	Варианты
Объем изменений	Нет
	Связанные требованием атрибуты (приоритет, обоснование источник и т. п.)
	Сама формулировка требования
	Связанная информация (тесты, ограничения дизайна, определения данных и т. п.)
Механизм повторного использования	Копирование и вставка из другой спецификации
	Копирование из библиотеки повторно используемых требований
	Обращение к исходному источнику

Объем повторного использования

Первое измерение определяет объем материала, который можно использовать повторно. Можно повторно использовать одно функциональное требование. Или его формулировку вместе со всеми связанными относящимися к целевому проекту атрибутами, такими как обоснование, источник, приоритет и другими. В некоторых случаях можно повторно использовать не одни только требования, но связанные артефакты: определения данных, приемочные тесты, соответствующие бизнес-правила, ограничения, предположения и т. п. Часто можно повторно использовать набор связанных требований, например все функциональные требования, относящиеся к определенной функции. В приложениях, работающих на похожих платформах, например на различных операционных системах смартфонов, можно повторно использовать требования и элементы дизайна, но вряд ли очень много кода.

В идеальном случае можно повторно использовать целый пакет требований, моделей, элементов дизайна, код и тесты. То есть вы повторно используете практически без изменений целый раздел реализованной функциональности из связанного продукта. Такой уровень повторного использования бывает, когда общие операции используются во многих проектах на общей платформе. Примерами таких операций могут быть приемы обработки ошибок, внутреннее журналирование данных и отчетность, абстракции коммуникационных протоколов и справочные системы. Эти функции должны разрабатываться для повторного использования с помощью четких API-интерфейсов вместе со всей вспомогательной документацией и артефактами тестирования.

Рассказ об успешном повторном использовании

Мне как-то пришлось работать для крупной компании розничной торговли, в которой нужно было объединить два интерактивных каталога — для потребителей и компаний — в одной новой системе. Бизнес-цель за-

ключалась в снижении затрат на обслуживание и упрощении добавления новых функций, которые должны присутствовать в обоих каталогах. В первую очередь мы разработали требования к потребительскому каталогу, основываясь на функциональности существующего каталога. Перейдя к корпоративному каталогу, мы взяли за основу эти же требования к потребительскому каталогу и просто скорректировали их. В новом корпоративном каталоге были и новые требования. Проект был реализован в запланированные сроки отчасти благодаря экономии времени из-за повторного использования.

Объем изменений

Следующий тип — сколько изменений нужно внести, чтобы существующие требования можно было использовать в новом проекте. В некоторых случаях вы сможете повторно использовать требование без изменений. В приведенном ранее примере авиакомпании многие функциональные требования одинаковы для терминала и веб-сайта, предоставляющего возможность пассажирам регистрироваться на рейс. В других случаях вы сможете повторно задействовать формулировку требования, но потребуется изменить ряд его атрибутов, таких как приоритет или основание в том виде, в котором оно применяется к новой системе. Часто вы будете начинать с существующего требования, но изменять его в соответствии с реалиями нового проекта. Наконец, при изменении требования может потребоваться изменить часть дизайна и тестов, например при переносе функциональности с компьютера на планшет, оборудованный сенсорным экраном, а не мышью и клавиатурой.

Механизм повторного использования

Самая элементарная форма повторного использования — простое копирование части требований из другой спецификации или библиотеки повторно используемых требований. Хранить информацию об источнике исходного требования не нужно, и копию требования можно изменять, как заблагорассудится. Копирование и вставка в проекте увеличивает объем спецификаций из-за дублирования информации. Каждый раз, когда вы копируете и вставляете спецификацию, вы должны задуматься. Так же, как при копировании кода, копирование и вставка требований может создавать проблемы, из-за несоответствия контекста, потому что он не переносится вместе с операцией вставки.

В большинстве случаев повторное использование существующей информации лучше выполнять ссылаясь, а не реплицируя ее. Это означает, что исходный источник информации должен быть доступным любому, кому может потребоваться посмотреть на требование, и должен сохраняться на том же месте. Если вы храните свои требования в документе и хотите, чтобы они отображались в нескольких местах, можете воспользоваться функцией перекрестных ссылок в своем редакторе, вставив ссылки на основной экземпляр

требования (Wiegers, 2006). При изменении основного экземпляра все изменения моментально переносятся во все копии, связанные перекрестными ссылками. Это позволяет избежать рассогласования, когда один экземпляр меняется вручную, а остальные остаются без изменений. Вместе с тем, сохраняется риск, что все требования изменяться в одночасье, если кто-то может изменить основной экземпляр.

Еще одни способ «копирования в ссылке» — хранить не саму информацию требования, а просто указатель на него в проектной документации. Допустим, вам надо повторно использовать описания некоторых пользовательских классов из других проектов в вашей организации. Первым делом соберите такую повторно используемую информацию в общем месте. В этом наборе могут быть документы, электронные таблицы, HTML- или XML-файлы, базы данных или специализированные средства работы с требованиями. Присвойте каждому объекту в таком наборе уникальный идентификатор. Чтобы вставить эту информацию по ссылке, вставьте идентификаторы повторно используемых объектов в соответствующий раздел документа. Если позволяет технология, включите гиперссылку непосредственно на повторно используемый объект в наборе информации. Пользователю, заинтересовавшемуся описанию такого класса, будет достаточно щелкнуть ссылку, чтобы перейти к основному источнику. Если вы надлежащим образом храните этот набор повторно используемых артефактов, ссылки и целевая информация всегда будет актуальной.

Намного более эффективный способ повторного использования — хранить требования в средстве управления требованиями, как описано в главе 30. При наличии соответствующей функциональности можно повторно использовать требование, которое уже есть в базе данных, не реплицируя его. Некоторые из таких средств хранят хронологию версий отдельных требований, что позволяет повторно использовать определенную версию требования или набор связанных требований. Если кто-то изменит требование в базе данных, используемая вами более старая версия останется. Можно создать собственную версию требования в соответствии с потребностями вашего проекта, не препятствуя другим повторно использовать это требование.

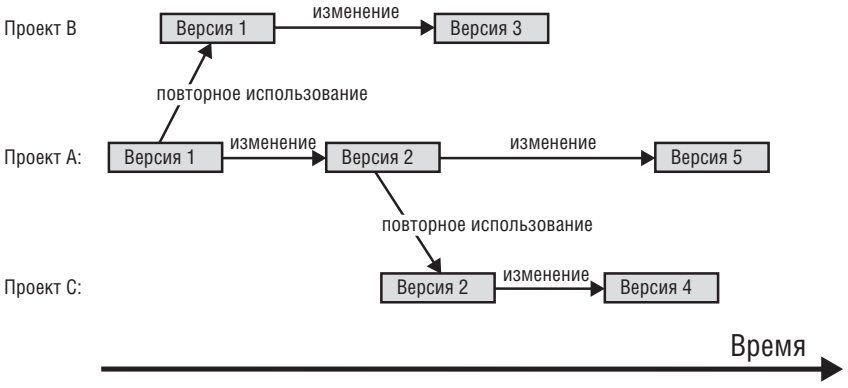


Рис. 18-1. Развитие требования при его использовании в нескольких проектах

Этот процесс показан на рис. 18-1. В проекте А создается начальная версия требования. Позже это требование повторно используется в проекте В, так что в этих проектах применяется одна версия. После этого в проекте А требование меняется, и появляется версия 2. Но версия 1 осталась в проекте В. Если позже в проекте В потребуется изменить требование, будет создана версия 3, которая никак не повлияет на остальные проекты, в которых используется это же требование.

Типы информации требований, поддающихся повторному использованию

В табл. 18-2 перечислены некоторые из связанных с требованиями активов, которые обычно поддаются повторному использованию в разных областях применения. Некоторые из этих активов относятся к нескольким областям применения. У некоторых типов активов очень широкие возможности повторного использования, в частности это относится к требованиям к поддержке специальных возможностей (часть удобства использования).

Набор связанных требований в определенной функциональной области обладают большим потенциалом повторного использования, чем одиночные, изолированные требования. Один пример — безопасность (Firesmith, 2004). Нет смысла каждой проектной команде в организации каждый раз заново изобретать требования к имени пользователя и аутентификации, изменению и сбросу паролей и т. п. Если вы можете написать набор исчерпывающих, качественно сформулированных требований для этих стандартных функций, их можно многократно использовать снова, экономя время и обеспечивая единообразие всех приложений. Может быть возможным повторно использовать наборы ограничений в определенной рабочей среде или платформе. Например, разработчики приложений для смартфонов должны знать размер экрана, разрешение и ограничения процессов взаимодействия с пользователем. Далее приводятся некоторые другие группы связанных требований, которые можно повторно использовать целыми наборами:

- функциональность и связанные исключения и приемочные тесты;
- объекты данных и соответствующие им атрибуты и проверки;
- связанные с выполнением нормативов и предписаний бизнес-правила, такие как закон Сарбейнса-Оксли (Sarbanes-Oxley, SOX), другие отраслевые ограничения, а также ориентированные на выполнение политик в организации директивы;
- симметричные пользовательские функции, такие как отмена и повтор правки (если повторно используется функция отмены правки в приложении, можно также копировать соответствующие требования к повторной правке);
- связанные операции, выполняемые с объектами данных, такие как создание, чтение, обновление и удаление.

Табл. 18-2. Некоторые типы повторно используемой информации требований

Область повторного применения	Кандидаты активов требований на повторное использование
В продукте или приложении	Пользовательские требования, отдельные функциональные требования в вариантах использования, требования к производительности, требования к удобству использования, бизнес-правила
В рамках семейства продуктов	Бизнес-цели, бизнес-правила, модели бизнес-процессов, контекстные диаграммы, карты экосистемы, пользовательские требования, базовые функции продукта, профили заинтересованных лиц, описания классов пользователей, архетипы пользователей, требования к удобству использования, требования к безопасности, требования к соответствию нормативам и законам, сертификационные требования, модели и определения данных, приемочные тесты, словарь терминов
В рамках предприятия	Бизнес-правила, профили заинтересованных лиц, описания классов пользователей, архетипы пользователей, требования к безопасности, словарь терминов
В рамках предметной области	Модели бизнес-процессов, функции продукта, пользовательские требования, описания классов пользователей, архетипы пользователей, приемочные тесты, словарь терминов, модели и определения данных, бизнес-правила, требования к безопасности, требования к выполнению нормативов и законов
В операционной среде или платформе	Ограничения, интерфейсы, инфраструктуры функциональности, необходимой для поддержки определенных типов требований (например, к генератору отчетов)

Популярные сценарии повторного использования

Создаете ли вы семейство продуктов, корпоративные приложения или даже продукт, с функцией, способной работать во многих контекстах, — всегда есть возможности повторного использования. Посмотрим на несколько сценариев, в которых бывает возможность повторного использования требований.

Семейства продуктов

Как правило при создании продуктов одного семейства много функциональности повторяется. Иногда создают различные варианты базового продукта для разных клиентов или рынков. Требования, созданные для одного варианта для конкретного клиента, может быть возможным добавить в общую спецификацию базового продукта. Другие семейства продуктов представляют собой набор связанных продуктов, базирующийся на общей платформе.

Например, поставщик популярного пакета подготовки декларации о доходах предлагает бесплатную интернет-версию, а для установки на компьютер предлагаются версии базовая, люксовая, домашняя, премиальная, а также бизнес-версия. Проанализируйте функции семейства программных продуктов на предмет функций, которые:

- общие для всех продуктов семейства;
- необязательные, то есть не во всех продуктах семейства;
- изменчивые, в различных продуктах семейства функциональность немного отличается (Gomaa, 2004; Dehlinger и Lutz, 2008).

Общие функции предлагают максимальные возможности повторного использования, и не просто отдельных требований, а всех последующих результатов работы по цепочке, включая компоненты архитектуры, дизайна, код и тесты. Это самая мощная форма повторного использования, но не всегда удается обнаружить возможность применить ее. Повторное использование общей функциональности намного лучше, чем каждый раз повторно реализовывать ее, возможно каждый раз немного меняя без особых на это причин. Надо обращать внимание на любые ограничения, которые операционная среда или аппаратная платформа определенных продуктов может налагать на возможности повторного использования. Если в разных продуктах семейства реализация должна отличаться, вы можете повторно использовать только требования, но не дизайн или код.

Реинжиниринг и замена системы

При реинжиниринге или замене системы всегда повторно используются часть требований из исходной версии системы, даже если эти «требования» никогда не излагались в письменном виде. Если вам приходится выполнять обратный инжиниринг знаний, воплощенных в старой системе, для выяснения возможности повторного использования, может потребоваться подняться на более высокий уровень абстрагирования, чтобы не погрязнуть в деталях реализации. Часто удается обнаружить бизнес-правила, реализованные в старой системе и повторно использовать их в будущих проектах, обновляя их по мере необходимости, например как в случае с нормативами и требованиями закона.

> **Внимание!** Не поддавайтесь искушению для экономии времени повторно задействовать слишком большую часть старой системы, упуская при этом возможности, предлагаемые платформами, новыми архитектурами и технологическими процессами.

Другие возможности повторного использования

В табл. 18-3 перечислены другие ситуации, в которых часто повторно используют информацию требований. Обнаружив любую из перечисленных возможностей в своей организации, подумайте, стоит ли собирать повторно используемые артефакты в общее хранилище и обеспечивать управление

этой информацией как общекорпоративным активом. Если вы ранее работали над похожим проектом, посмотрите, нельзя ли снова задействовать какие-то артефакты из предыдущего проекта.

Табл. 18-3. Обычные возможности повторного использования требований

Возможность повторного использования	Примеры
Бизнес-процессы	Часто бизнес-процессы бывают одинаковыми во многих организациях и должны одинаково обслуживаться программным обеспечением. Многие организации поддерживают набор описаний бизнес-процессов, которые можно использовать в разных ИТ-проектах
Распределенные среды	Часто одна система развертывается многократно с небольшими вариациями. Такая ситуация очень часто наблюдается в розничных магазинах и на складах. При развертывании всех экземпляров повторно используется общий набор требований
Интерфейсы и интеграция	Часто возникает потребность в повторном использовании требований к интерфейсам и процессам интеграции. Например, в больницах у большинства вспомогательных систем должны быть интерфейсы входного и выходного взаимодействия с системами приема, выписки и перевода. Это также верно в отношении финансовых интерфейсов системы управления ресурсами предприятия
Безопасность	Аутентификация пользователей и требования к безопасности часто одинаковы во многих системах. Например, у систем может быть общее требование, что все продукты должные поддерживать единый вход с использованием Active Directory для аутентификации пользователей
Общие функции приложений	Бизнес-приложения часто содержат общую функциональность, для которой можно повторно использовать требования, и даже всю реализацию целиком. Вот несколько примеров: операции поиска, печать, файловые операции, пользовательские профили, отмена и повтор операции и форматирование текста
Аналогичные продукты для разных платформ	Единый базовый набор требований используется даже при наличии некоторых подробных требований и/или отличий в дизайне пользовательского интерфейса, обусловленных платформой. В качестве примера можно привести приложения, которые работают в среде Mac и Windows или в iOS и Android
Стандарты, нормативы и правовые нормы	Во многих организациях разработаны наборы стандартов — часто основывающихся на нормативно-правовых актах, — которые определены как набор требований. Их можно повторно использовать между проектами. Примеры: Стандарты ADA для дизайна с расчетом на ограниченные возможности и требования закона об ответственности и переносе данных о страховании здоровья граждан (HIPAA),

Схемы требований

Использование знания, которое облегчает бизнес-аналитику написание требований можно считать повторным использованием. Такова аргументация в пользу схем требований: упаковать значительные объем знаний о конкретном типе требований так, чтобы бизнес-аналитику было удобно определять такие требования.

Впервые сформулированное Стивеном Уитхоллом (Stephen Withall, 2007), понятие *схема требований* (requirement pattern) предлагает систематический подход к определению конкретного типа требований. Схема определяет шаблон с категориями информации для каждого из популярных типов требований, которые могут использоваться в проекте. У различных типов схем требований собственные наборы категорий содержимого. Заполнение шаблона скорее всего позволит получить более подробную спецификацию требования, чем если бы бизнес-аналитик просто изложил суть естественным языком. Структура и содержание требования, написанного по схеме, облегчает повторное использование.

Схема требования состоит из нескольких разделов (Withall, 2007):

1. **Инструкция** Базовая информация о схеме, включая связанные схемы, ситуации, к которым она применима (или неприменима), а также информация о том, как писать требования данного типа.
2. **Содержимое** Подробное последовательное (одно за одним) объяснение сведений, которые данное требование должно содержать.
3. **Шаблон** Определение требования с полями подстановки, где должны размещаться различные части информации. Он может служить отправной точкой в стиле «заполните пробелы» для написания определенных требований данного типа.
4. **Примеры** Один или больше наглядных примеров требований данного типа.
5. **Дополнительные требования** Дополнительные требования, которые могут описывать определенные аспекты темы или объяснять, как писать наборы подробных требований, формулирующих, что нужно сделать для удовлетворения исходного высокоуровневого требования.
6. **Особенности разработки и тестирования** Обстоятельства, которые должны учитывать разработчики при реализации требования данного типа, и обстоятельства, которые должны учитывать тестировщики при тестировании таких требований.

В качестве примера возьмем функцию создания отчетов, которая присутствует во многих приложениях. Уитхолл (Withall, 2007) предоставляет схему для спецификации требований, определяющих отчеты. Эта схема включает шаблон, показывающий, как структурировать различные элементы отчета в набор более подробных требований, составляющих полную спецификацию отчета. Но шаблон всего лишь часть схемы. Схема также содержит пример

требования к отчетности, возможные включаемые требования и объемные инструкции по определению, реализации и тестированию этих требований.

Вы можете самостоятельно создавать схемы требований, идеально соответствующие принятым в организации стилю и существующим проектам. Следование схеме помогает обеспечить единообразие, а также создавать более насыщенные и точные требования. Простые похожие на этот шаблоны напоминают вам о важной информации, о которой вы можете забыть. Если вам нужно написать требование на незнакомую тему, схема значительно ускорит процесс и вам потребуется меньше тратить время на самостоятельные исследования.

Средства, облегчающие повторное использование

В идеальном мире организация хранила бы все требования к ПО в средстве управления требованиями с полным набором связей между требованиями. Такие связи соединяют каждое требование с родительским требованием или другим источником, с требованиями, от которого зависит данное, и с относящимися к данному требованию артефактами разработки. В таком средстве доступны все предыдущие версии каждого требования. Это наилучший способ обеспечить эффективное повторное использование в большом масштабе во всем приложении, наборе продуктов или в целой организации.

Немногим организациям удалось достичь подобного уровня совершенства, но хранение требований в специализированном средстве все равно во многих отношениях способствует повторному использованию (Akers, 2008). Коммерческие средства управления требованиями предоставляют различные возможности по поддержке повторного использования. В некоторых есть даже крупные библиотеки требований, которые относятся к определенным предметным областям и готовы к повторному использованию. При выборе средства в рамках процесса оценки учитывайте свои ожидания относительно того, как это поможет повторно использовать требования. В главе 30 описаны типичные функции имеющихся на рынке средств управления требованиями.

Это средство может обеспечить повторное применение определенного требования за счет совместного его использования во многих проектах и определения основной версии требований. При повторном использовании вы должны думать о том, что произойдет, если вы измените исходное требование или его копии. Некоторые средства позволяют заблокировать содержимое, чтобы можно было редактировать только исходный экземпляр требования. Это гарантирует, что при редактировании требования оно обновляется во всех местах, где оно используется повторно. Естественно если вы начнете с повторно использованного требования и захотите изменить его в соответствии с потребностями конкретного проекта, такая жесткая связь вам ни к чему.

В этом случае желательно дублировать требование в режиме, который позволит изменять копию требования.

Аналогично, при копировании требования, у которого есть связи с другими требованиями или артефактами, может потребоваться решить, надо ли переносить все эти связи в копию. Иногда в новый проект нужно взять только само требование вместе с его дочерними требованиями и требованиями, от которых оно зависит. Обычно такое бывает, когда функция остается такой же, но меняются платформы, как в случае приложений, работающих в веб-браузере, на планшете, смартфоне и терминале.

> **Внимание!** Если бизнес-аналитик не находит нужного в хранилище повторного использования, совершенно неважно, насколько качественны хранящиеся в нем требования или сколько времени они могут сэкономить, — бизнес-аналитику придется писать требования с нуля. Создание повторно используемых требований по заданным схемам дает набор полей, по которым можно искать нужное требование. Некоторые люди ратуют за добавление осмысленных ключевых слов или атрибутов требований, которые могут облегчить поиск.

Как сделать требования повторно используемыми

Одного существования требования недостаточно, чтобы его можно было повторно использовать. Оно может быть тесно привязано к конкретному проекту. Или оно может быть написано на слишком высоком уровне абстракции, потому что бизнес-аналитик предполагает наличие определенных знаний у разработчиков или потому что некоторые подробности передавались только устно. В требовании может отсутствовать информация об обработке возможных исключений. Может потребоваться скорректировать исходные требования, чтобы повысить их ценность для будущего использования.

Качественно написанные требования хорошо поддаются повторному использованию. Усилия по обеспечению возможности повторного использования требований также увеличивают их ценность для проекта, в котором они изначально были созданы — они просто становятся более качественными. При повторном использовании нужно знать о зависимостях между требованиями, а также о связанных требованиях, которые также могут поддаваться повторному использованию. *Повторное использование* экономит время и деньги, но превращение чего-то в *повторно используемый компонент* скорее всего требует времени и денег.

Повторно используемые требования должны быть написаны с правильным уровнем абстракции и областью действия. Относящиеся к предметной области требования отличаются низким уровнем абстракции. Они скорее всего найдут применение только в их исходной предметной области (Shehata, Eberlein и Hoover, 2002). У общих требований больший потенциал повторного использования в разных системах. Однако если их попытаться исполь-

зовать на слишком общем уровне, вы мало выиграете, потому что бизнес-аналитику все равно придется выяснять детали. Сложно найти правильный баланс между упрощением для повторного использования (с большим уровнем абстракции и обобщенными требованиями) и получением выигрыша от него (для более подробных и узкоспециализированных требований).

На рис. 18-2 показан пример. Представьте, что вы создаете приложение, в котором есть пользовательское требование, предусматривающее возможность принимать оплату с помощью кредитных карт. Это пользовательское требование разбивается на ряд связанных функциональных и нефункциональных требований к обработке платежей с помощью кредитных карт. В других приложениях также может потребоваться поддержка кредитных карт, так что у этого набора требований хорошие перспективы на повторное использование.

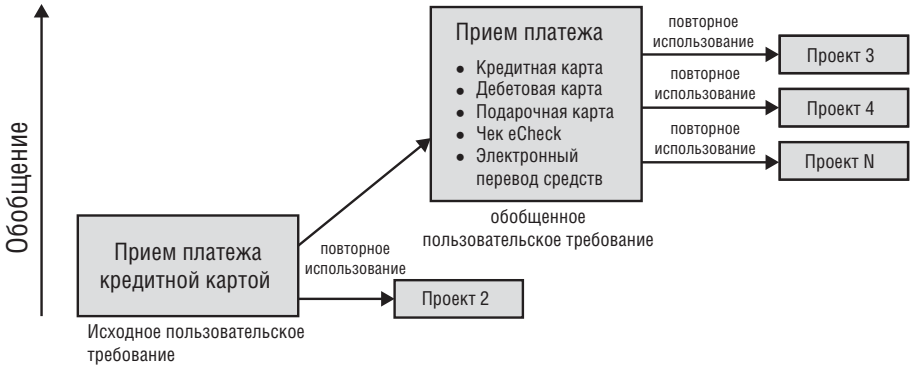


Рис. 18-2. У обобщенного требования больше потенциал повторного использования

Представьте, что вы можете обобщить это пользовательское требование, чтобы охватить несколько способов платежа: кредитные, дебетовые и подарочные карты, а также eCheck и электронный перевод средств. Полученное в результате требование предоставляет больше возможностей повторного использования в более широком диапазоне будущих проектов. В одном проекте может требоваться только процессинг кредитовых карт, а в других могут потребоваться несколько способов обработки платежей. Обобщение исходного пользовательского требования — от «прием платежа по кредитной карте» до «прием платежа» — может оказаться полезным даже в текущем проекте. Даже если клиент изначально просил обеспечить обработку платежей с помощью кредитных карт, пользователи могут потребовать использовать многие виды платежей — сейчас или в будущем.

Выбор подходящего уровня абстракции для требования может также окупиться во время разработки. В одном проекте, где была именно такая потребность в поддержке многих способов оплаты, создание четких требований и правил на каждый случай позволило обнаружить как сходные черты, так и отличия. Независимо от возможностей повторного использования в будущем, более высокий уровень абстракции облегчает дизайн и разработку.

Это положительная сторона. А отрицательная заключается в том, что для обобщения исходного требования потребуются определенные усилия. Это инвестиция, которую вы делаете в повторное использование в надежде, что она окупится и даже принесет прибыль за счет многократного повторения в будущих экземплярах. Вам решать — просто разместить имеющиеся требования в общее хранилище в расчете на возможное повторное использование в будущем или потратить силы на совершенствование требования, чтобы оно хорошо поддавалось повторному использованию в будущих проектах.

Взрывообразное размножение «повторно используемых требований»

Коллега рассказал о том, как можно снизить потенциал повторного использования требования за счет излишней его детализации. В команде, которой было поручено написание требований, были одержимы повторным использованием. Бизнес-аналитик посчитал, что если задокументировать подробности каждого требования отдельно, тогда их можно будет повторно использовать. В результате у них получилось 14 тысяч требований! В хранилище были записи, которые должны бы быть одним требованием, но были структурированы как одно родительское и несколько дочерних требований, каждое из которых дающее подробности относительно родителя. Таким образом, детализированные требования имели смысл только в этом одном приложении.

Такой объем требований также сильно затруднил цикл тестирования, что привело к ежедневным жалобам тестировщиков. Им приходилось тратить намного больше времени на написание тестов, потому что требовалось обработать такое огромное число требований. Тестировщики должны были указывать идентификаторы требований, чтобы можно было отслеживать покрытие требований тестами, но большим числом связей с огромным числом требований стало очень трудно управлять. Кроме того требования претерпели экстенсивные изменения: они никогда не были полностью стабилизированы. Все эти обстоятельства привели к тому, что проект завершился с задержкой на год и так и не позволил получить желаемый набор повторно используемых требований.

Препятствия и факторы успеха повторного использования требований

Повторное использование требований кажется замечательной идеей, но оно не всегда практично или уместно. В этом разделе описываются некоторые соображения, которые могут позволить вашей организации успешно реализовать повторное использование требований.

Препятствия для повторного использования

Первый шаг на пути к преодолению препятствия — распознать и понять его. Вот несколько преград, которые вам могут встретиться на пути к повторному использованию требований.

Отсутствующие или низкокачественные требования Обычное препятствие заключается в том, что в предыдущем проекте требования не были задокументированы, поэтому повторное их использование невозможно. И даже если вы найдете подходящее требование, оно может быть некачественно написано, неполно или плохо соответствовать вашему проекту. Требование может быть задокументировано, но исходное требование к старому приложению не поддерживалось в актуальном состоянии по мере развития приложения, что привело к безнадежному устареванию требования.

Синдромы NIH и NAH NIH расшифровывается как «not invented here», то есть «изобретено не здесь». Некоторые люди с неохотой используют требования из другой организации или обобщенные требования из открытых источников. Требования, написанные в другом месте, может быть, сложно понимать: может отличаться терминология, требования могут ссылаться на недоступные документы, контекст исходных требований может не поддаваться определению, а также может отсутствовать объяснение важной общей информации. Бизнес-аналитик может разумно решить, что на написание требований с нуля потребуется меньше усилий, чем на то, чтобы разобраться и скорректировать существующие требования.

Синдром NAH («not applicable here», то есть «здесь неприменимо») проявляется в том, что люди противятся новому процессу или подходу, говоря, что он неприменим к их проекту или организации. Они утверждают, что «у нас все иначе». Участники проекта могут считать его уникальным, поэтому никакие существующие требования к нему не подойдут. Иногда это верно, но часто формулировки NIH и NAH свидетельствуют об отсутствии гибкости.

Стиль письма Бизнес-аналитики в предыдущих проектах могут использоваться широкое разнообразие приемов и соглашений о представлении требований. Лучше всего принять некоторые стандартные условные обозначения для документирования требования, чтобы упростить их повторное использование, например можно применить схемы. Если требования написаны примерно на одном уровне детализации, бизнес-аналитику проще искать подходящие требования на соответствующем уровне. Также важно единство терминологии. Вы можете пропустить подходящее для повторного использования требование, просто потому, что его терминология отличается от той, которую используют ваши заинтересованные лица. Требования на естественном языке отличаются особенной двусмысленностью, пробелами информации и скрытыми предположениями. Эти проблемы снижают возможности повторного использования.

Требования с внедренными ограничениями дизайна предоставляют немного вариантов для использования в другой среде. Вспомните о терминале

регистрации в аэропорту. Если подробности пользовательского интерфейса киоска внедрить в требование, его нельзя будет использовать при разработке программы практически с такой же функциональностью, но работающей на веб-сайте.

Непоследовательная организация Бывает часто сложно найти требования для повторного использования, потому что авторы организуют свои требования различным образом: по проектам, по потокам процессов, по бизнес-подразделениям, по функциям продукта, по категориям, по подсистемам или компонентам и т. п.

Тип проекта У требований, тесно связанных с определенными средами или платформами реализации, меньший потенциал повторного использования. В быстро развивающихся предметных областях еще нет достаточного повторного использования набора требований; актуальные сегодня требования могут завтра безнадежно устареть.

Право собственности Еще одно препятствие — право собственности на интеллектуальный продукт (Somerville и Sawyer, 1997). При разработке программного продукта для конкретного клиента требования скорее всего становятся исключительной интеллектуальной собственностью этого клиента. У вас может не быть юридических прав повторно использовать любое из этих требований в других разрабатываемых вами системах для своей компании или других клиентов.

Факторы успеха повторного использования

В организации, где серьезно относятся к повторному использованию, должны создать механизмы, облегчающие повторное использование и применение существующей информации. Это означает, извлечение поддающейся повторному использованию информации из конкретного проекта, чтобы другие могли получить доступ и задействовать ее. Ниже приведены советы по успешному повторному использованию.

Хранилище Нельзя повторно использовать что-то, что нельзя найти. Таким образом, требуется инструмент, обеспечивающий эффективное широкомасштабное повторное использование и представляющий собой хранилище информации требований с возможностью поиска в нем. Это хранилище может принимать несколько форм:

- одна сетевая папка, где хранятся документы предыдущих требований;
- набор требований, который размещен в средстве управления требованиями и в котором можно выполнять поиск по всем проектам;
- база данных, хранящая наборы поддающихся повторному использованию требований, отобранных из проектов и дополненных ключевыми словами, позволяющими бизнес-аналитикам определять источник требований, судить об их уместности и ограничениях.

Представьте, что вы поручили кому-то управлять хранилищем повторно используемых требований. Этот человек должен по мере необходимости

адаптировать существующие знания требований для представления и хранения в форме, подходящей для эффективного поиска, извлечения и повторного использования. Для работы с повторно используемыми требованиями можно приспособить схему, похожую на ту, что используется для хранения и управления бизнес-правилами как корпоративным активом.

Качество Никому не хочется тащить в проект старый мусор. Тот, кто пытается повторно задействовать требование, должен быть уверен в его качестве. И даже если повторно используемое требование неидеально, нужно попытаться улучшить его, когда оно применяется повторно. Таким образом вы сможете постепенно улучшить требование, со временем повысив его пригодность к повторному использованию в будущих проектах.

Взаимодействия У требований часто есть логические ссылки или зависимости между собой. Использование связей — средство, позволяющее определять эти зависимости, чтобы люди знали, во что ввязываются, выбирая то или другое требование для повторного использования. Повторно используемые требования должны соответствовать существующим бизнес-правилам, ограничениям, стандартам, интерфейсам и ожиданиям по качеству.

Терминология Создание общей терминологии и определений во всех ваших проектах будет полезным для повторного использования. Вариации в терминологии не закрывают путь к повторному использованию требований, но нужно следить за несоответствиями и предпринимать дополнительные усилия для предотвращения недопонимания. Словари терминов и данных — хорошие источники повторно используемой информации. Вместо того, чтобы вставлять весь словарь терминов в каждую спецификацию требований, создайте ссылки от ключевых терминов к их определению в общем словаре.

Организационная культура Руководство должно способствовать повторному использованию, поощряя два вида деятельности: добавление высококачественных компонентов с реальным потенциалом повторного использования и эффективное повторное использование существующих артефактов. Отдельные сотрудники, проектные команды и организации, в которых практикуется повторное использование, скорее всего отличаются повышенной производительностью. Если есть культура повторного использования, перед созданием собственных требований бизнес-аналитик изучает хранилище повторно используемых требований. Он начинает с пользовательской истории или другого высокоуровневого требования и смотрит, насколько он может повторно задействовать имеющуюся подробную информацию.

Проектные требования являются ценной информацией организации. Чтобы получить максимум отдачи от разработки требований, попробуйте определить те знания о требованиях, которые можно считать общекорпоративным активом. Повторно используемые требования не должны быть идеальными, чтобы приносить реальную пользу. Даже если они позволят сэкономить всего лишь 20% работы, это уже серьезный выигрыш и экономия времени, которое было бы потрачено на написание требований с нуля. Культура поощрения бизнес-аналитиков сначала заимствовать и лишь затем

создавать новое и небольших дополнительных инвестиций в обеспечение повторного использования требований позволяет повысить продуктивность как аналитиков, так и разработчиков, а также способствует созданию высококачественных продуктов.

Что дальше?

- Изучите свой текущий проект на предмет того, можно ли упростить наборы требований за счет повторного использования знаний требований из предыдущих проектов или других источников.
- Проанализируйте свой текущий проект на предмет обнаружения требований, которые поддаются повторному использованию. При оценке возможностей повторного использования проанализируйте каждое требование в соответствии с табл. 18-2. Помните, что у вас должен быть реальный шанс экономии средств за счет извлечения повторно используемых активов, упаковки, хранения и предоставления доступа к ним других людей, в противном случае не стоит тратить силы.
- Подумайте, какую информацию о повторно используемых требованиях нужно хранить, чтобы облегчить бизнес-аналитикам в будущем поиск и определение, подходят ли требования к их проектам. Выберите хранилище, в котором будут размещаться требования для повторного использования.

Глава 19

От разработки требований — к следующим этапам

Куратор проекта системы Chemical Tracking System Жерар сомневался по поводу необходимости тратить много времени на определение требований. Тем не менее он присоединился к разработчикам и другим сторонникам продукта, которые проводили однодневный тренинг, посвященный работе над требованиями к ПО. Тренинг убедил Жерара поддержать работу над требованиями.

По мере развития проекта Жерар получил отличные отзывы от представителей пользователей о том, насколько хорошо прошла разработка требований. Он даже организовал ланч для аналитиков и сторонников продукта, чтобы отпраздновать создание основной версии требований для первого выпуска системы. На ланче Жерар поблагодарил тех, кто занимался сбором информации для создания требований, за их вклад и коллективную работу. А затем он сказал: «Теперь, когда с требованиями все в порядке, я с нетерпением ожидаю скорого появления готового продукта».

«Мы еще не готовы писать код продукта, — ответил менеджер проекта. — Мы планируем выпускать систему поэтапно, по одному выпуску каждые два месяца. Если сейчас мы потратим еще немного времени на дизайн, разработчикам будет проще в дальнейшем добавлять функциональность. По мере продвижения вперед мы также будем узнавать больше о требованиях. Тем не менее в каждом выпуске вы будем показывать тебе частично работающее ПО».

Жерар расстроился. Все выглядело так, как будто разработчики сознательно медлили, вместо того чтобы сразу приступить к программированию. Однако не делал ли он преждевременных выводов?

Опытные менеджеры проектов и разработчики понимают ценность преобразования требований к ПО в надежный дизайн и рациональные планы. Они необходимы в тех случаях, когда следующая версия представляет как 1, так и 100% всего конечного продукта. В этой главе мы рассмотрим, как преодолеть пропасть, которая отделяет разработку требований от успешного выпуска продукта. Часть из этих действий относятся к области ответственности бизнес-аналитика, а остальные — ответственность менеджера проекта. Мы обсудим несколько вариантов влияния требований на планы проекта, ди-

зайн, код и тестирование, как показано на рис. 19-1. Помимо этих отношений существует связь между требованиями к будущему ПО и другие проектные требования и требования по переходам. К ним относятся миграция данных, проектирование обучения и поставки, изменения бизнес-процессов и организационные изменения, модификация инфраструктуры и другие. Эти операции в этой книге не обсуждаются.

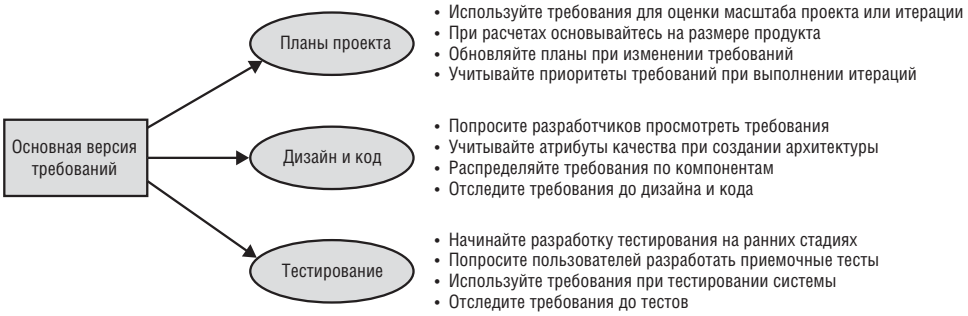


Рис. 19-1. Влияние требований на планирование проекта, дизайн, написание кода и тестирование

Оценка объема работ по реализации требований

На самых ранних стадиях планирования проекта нужно оценить, какая часть графика и работ по проекту будет посвящено определению требований. Карл Вигерс (Karl Wiegers, 2006) предлагает несколько способов оценки и ряд факторов, которые могут приводить как увеличению, так и уменьшению времени по сравнению с предварительной оценкой. В небольших проектах обычно от 15 до 18% всех затрат приходятся на разработку требований (Wiegers, 1996), однако показатель зависит от объема и сложности проекта. Несмотря на опасения, что работа над требованиями замедлит создание продукта, доказано, что понятные требования ускоряют процесс разработки, что и показывают следующие примеры:

- изучение 15 проектов в сфере телекоммуникаций и банковской сфере показало, что в наиболее успешных проектах примерно 28% ресурсов тратилось на разработку, моделирование, утверждение и проверку требований (Hofmann и Lehner, 2001). На разработку требований для среднего проекта необходимо 15,7% всех ресурсов и 38,6% времени;
- в проектах NASA, в которых затрачивалось более 10% всех ресурсов на разработку требований, затраты и отклонения от графика оказались существенно ниже, чем в проектах, где на требования затрачивалось меньше ресурсов (Hooks и Farry, 2001);

- исследования, проводимые в Европе, показали, что команды, разрабатывающие продукты более быстро, посвятили больше времени и усилий требованиям, чем более медленные команды (табл. 19-1) (Blackburn, Scudder и Van Wassenhove, 1996).

Табл. 19-1. Затраты на требования ускоряют разработку

	Усилия, затраченные на требования	**Время, затраченное на требования**
Более быстрые проекты	14%	17%
Более медленные проекты	7%	9%

Деятельность по разработке требований по-разному распределяется на протяжении всего проекта в зависимости от используемой в нем модели жизненного цикла — водопадной, итерационной или инкрементальной, как показано на рис. 3-3 в главе 3.

> **Внимание!** Старайтесь избегать паралича аналитического процесса. Если в самом начале проекта масса усилий тратится на разработку совершенных и полных требований — «раз и навсегда», то зачастую мало полезной функциональности удается реализовать в срок. С другой стороны, не следует избегать разработки требований вообще только из-за боязни паралича аналитического процесса. Как всегда это бывает в жизни, точка разумного баланса лежит где-то между этими крайними точками.

При оценке усилий, которые в проекте придется потратить на разработку требований, используйте собственный опыт. Обратитесь к затратам усилий на работу с требованиями в предыдущих проектах и оцените, насколько эффективной была работа над требованиями в тех проектах. Если вам удастся связать дефекты с низким качеством требований, наверное стоит больше поработать над требованиями. Естественно, что при такой оценке предполагается, что сохранились некоторые исторические данные о предыдущих проектах, на основании которых вы сможете лучше оценить будущие проекты. У вас может не быть таких данных, но если члены команды будут регистрировать, на что они тратят свое время в текущем проекте, эта «историческая информация» пригодится вам в будущих проектах. В этом нет ничего сложного. Запись оценки усилий и реальных затрат позволяет вам подумать над тем, как вам улучшить оценку в будущем.

В компании Seilevel, оказывающей консалтинговые услуги по требованиям, (это компания, где работает Джой) разработали эффективную методику оценки усилий по разработке требований в проекте, уточненную на основе рабочих оценок и фактических результатов многих проектов. В этой методике используются три взаимодополняющих оценки: процент общей работы, отношение «разработчик-бизнес-аналитик» и разбиение действий, в котором при получении оценки «снизу вверх» используются базовые затраты. Сравнение результатов всех трех оценок и согласование значительных расхождений позволяет бизнес-аналитикам получить наиболее точные оценки.

Первая оценка основана на проценте оцененной общей работы по проекту. В частности, мы полагаем, что на работу над требованиями приходится примерно 15% всех затрат по проекту. Это значение согласуется с цифрой, озвученной ранее в этом разделе. Так что если оценка общих затрат на проект составляет 1000 часов, то затраты на разработку требований оцениваются в 150 часов. Естественно, что общая оценка затрат на проект может меняться в процессе уточнения требований.

Вторая оценка содержит типичное отношение числа разработчиков к бизнес-аналитикам. По умолчанию мы используем отношение 6:1, то есть один бизнес-аналитик создает количество требований, достаточное чтобы загрузить работой шесть разработчиков. Бизнес-аналитики также будут взаимодействовать с тестировщиками, менеджером проекта и самим заказчиком, так что эта оценка охватывает всю работу бизнес-аналитика в проекте. В проектах по созданию тиражируемых решений это отношение меняется на 3:1 (три разработчика на одного бизнес-аналитика). Все еще есть много работы по выявлению требований по выбору, конфигурации и переходам, но команда разработчиков меньше, потому что большая часть кода приобретается, а не разрабатывается с нуля. Так что, зная размер команды разработчиков, можно оценить потребности в бизнес-аналитиках. Это всего лишь эмпирическое правило, а не отлитый в железобетоне закон на будущее, поэтому корректируйте эту цифру в соответствии с реалиями своей организации и типом проекта.

Третья оценка учитывает различные действия, выполняемые бизнес-аналитиком, на основе оценки числа различных артефактов, которые могут создаваться в конкретном проекте. Бизнес-аналитик может оценить число рабочих потоков, пользовательских историй, экранных форм, отчетов и т. п. и на основе этого сделать разумное предположение о том, сколько потребуется других артефактов требований. На основе оценок времени на выполнение отдельных действий, которые мы накопили в процессе работы над многими проектами, мы можем создать общую оценку затрат на создание требований.

Input					
Quantity	Items		Quantity	Items	
20	Existing pages of documentation for review		$ 750,000	Total project budget	
0	Existing systems being updated or replaced		$125	BA blended hourly cost	
5	Stakeholders		Standard	Type of project	
1	Interfacing Systems - small systems		5	Number of developers	
1	Interfacing Systems - medium systems		No	Is your team remote?	
1	Interfacing Systems - large systems		52	Project duration? (weeks)	
8	Process Flows		20	Requirements work duration? (weeks)	
2	BDDs				
8	Screens				
1	Reports				
Summary Total Effort Comparison					
		% of total project	Ratio of dev to BAs	Activity based	
	Number of BAs	1	0.8	0.8	*Excludes time off
	BA budget for requirements work	$ 113,000	$ 83,000	$ 77,000	
	BA budget for project duration	$ 293,000	$ 217,000	$ 200,000	

Рис. 19-2. Фрагмент результата таблицы для оценки затрат на требования

Мы создали средство для вычисления всех трех оценок работы над требованиями в виде электронной таблицы, которая есть в прилагаемом к этой книге материале. Рис. 19-2 иллюстрирует часть результатов этой таблицы. В разделе Summary Total Effort Comparison показаны оценка числа бизнес-аналитиков и бюджет на бизнес-аналитиков при работе над требованиями и в целом в проекте. Эти оценки служат исходной точкой для пересмотра различий, переговоров о ресурсах и планирования потребностей бизнес-аналитиков в проекте.

У таблицы оценки затрат на требования три листа. Первый содержит сводку, в которой надо ввести несколько характеристик проекта. Таблица оценивает различные элементы трех типов оценок. На втором листе находятся предположения, которые можно скорректировать в соответствии со своими предпочтениями. На третьем листе вы найдете инструкции, как использовать это средство оценки.

Имеющиеся в данном средстве предположения основаны на обширном практическом опыте работы с проектами компании Seilevel. Вам скорее всего потребуется изменить предположения для своей организации. Например, если ваши бизнес-аналитики новички или особенно высококвалифицированные специалисты, часть ваших оценок времени на отдельные действия могут отличаться от значений по умолчанию. Чтобы подогнать средство под реалии вашей компании, соберите данные своих проектов и в соответствии с ними скорректируйте параметры.

> **Внимание!** Все оценки основываются на знании и опыте того, кто оценивает, на момент, когда он делает свои предположения. Предварительные оценки, основанные на ограниченной информации, характеризуются высокой неопределенностью. Уточняйте свои оценки по мере накопления знаний и завершения части работы во время реализации проекта. Записывайте свои предположения, чтобы было ясно, как вы мыслили, формулируя свои цифры.

Бетти загнали в угол

Шридхар, менеджер многомиллионного проекта, предложил бизнес-аналитику Бетти обсудить ее начальную оценку того, сколько времени займет разработка требований. В процессе предыдущей почтовой переписки она оценила это время в восемь недель. Шридхар спросил: «Бетти, неужели тебе нужно целых восемь недель на сбор требований для нашего торгового портала? Я уверен, что твоя команда справится за четыре недели — система не настолько сложна. Нет, ну ведь люди приходят на вебсайт всего лишь, чтобы найти и купить нужный товар. И все! И вообще, менеджер разработчиков считает, что его команда в состоянии разработать систему вообще без требований, и именно это он собирается сделать, если требования не будут готовы через четыре недели».

Бетти зажата в угол. Ей остается только уступить и согласиться на нереальный четырехнедельный срок для этого крупного проекта. Или она

может настоять на своем, но рискует показаться неэффективной из-за предполагаемой «простоты» проекта. Вообще говоря, в настоящий момент Бетти не совсем уверена, сколько времени займет разработка адекватного набора требований, потому что не знает размер системы. Пока она не начнет анализ, она не узнает, сколько знаний ей не хватает.

Подобные ситуации — одна из основных причин, почему в компании Seilevel разработали описанное в этой главе средство оценки. Это средство Бетти использует в качестве союзника в этой непростой беседе со Шридхаром. Она может сказать: «Ну, если у меня только четыре недели, давай посмотрим, что я *смогу сделать* за это время». Бетти может эффективно очертить временные рамки разработки требований. Однако важно, чтобы Шридхар понимал, что понимание требований только видимой части айсберга неизбежно чревато неприятными неожиданностями в будущем в процессе выполнения проекта.

От требований — к планам проекта

Поскольку именно требования определяют запланированный объем работы по проекту, планы, сметы и графики следует разрабатывать на основе требований. Однако следует помнить, что наиболее важный результат проекта — это та система, которая соответствует бизнес-целям, а не обязательно та, в которой реализованы все требования согласно первоначальному плану проекта. В требованиях и планах указана первоначальная оценка стоимости затрат на проект, определенная членами команды. Однако границы проекты могут выйти за первоначальные рамки, да и первоначальные планы могут оказаться невыполнимыми или плохо соответствующими целям. Кроме того, бизнес-потребности, бизнес-правила и ограничения проекта могут меняться. Успех проекта сомнителен, если вы не будете обновлять ваши планы в соответствии с изменяющимися целями и обстоятельствами.

Оценка размера проекта и объема усилий на основе требований

Получение реалистичных оценок усилий и времени, необходимых для реализации проекта, зависит от многих факторов, но все начинается с оценки размера продукта, который планируется создать. Базовый размер проекта можно оценить на основе функциональных требований, пользовательских историй, моделей анализа, прототипов или элементов пользовательского интерфейса. Хотя не существует идеального мерила размера ПО, чаще всего используются следующие:

- количество отдельно тестируемых требований (Wilson, 1995);
- баллы функций (Jones, 1996b; IFPUG, 2010);

- баллы историй (Cohn, 2005; McConnell, 2006) или вариантов использования (Wiegers, 2006);
- количество, тип и сложность элементов графического пользовательского интерфейса;
- оценка строк кода, необходимого для реализации конкретных требований.

Какой бы вы метод не выбрали, используйте собственный опыт и природу разрабатываемого ПО. Понимание того, что команда добилась успеха при разработке схожих проектов, используя аналогичные технологии, поможет вам оценить результативность работы команды. Как только вы получите данные о размере, вы сможете оценить общие усилия, необходимые для реализации проекта. Усилия зависят от таких факторов, как размер команды (люди, выполняющие большое число задач, работают менее эффективно, а большое количество интерфейсов коммуникации замедляют работу) и запланированный график (плотный график увеличивает объем необходимых усилий).

Один подход заключается в использовании серийного инструмента для расчета, который предлагает допустимые комбинации затрат на разработку и сроков. Эти средства позволят вам скорректировать расчеты, выполненные на основе таких факторов, как квалификация разработчиков, сложность проекта, а также опыт команды в данной предметной области. Между размером продукта, затратами, сроком разработки, производительностью и временем становления команды существуют сложные нелинейные взаимоотношения (Putnam и Myers, 1997). Если вы разберетесь в них, вы сможете избежать «области невозможного», то есть такой комбинации факторов, зависящих от размера продукта, графика и сотрудников, при которой вероятность успеха исключительно мала.

В наилучших процессах оценки учитывается неопределенность на ранних этапах и постоянная изменчивость границ проекта. Те, кто использует такой процесс, выражают оценки в виде диапазонов, а не одного значения. Они управляют точностью своей оценки за счет расширения диапазона, основываясь на неопределенности и изменчивости данных, на которых базируется оценка.

В проектах гибкой разработки (agile) объем проектов измеряется в *баллах историй* (story point) — это мера относительных усилий, которые потребуются для реализации конкретных пользовательских историй. Оценка размера конкретной истории зависит от имеющихся — или отсутствующих — знаний о ней, ее сложности и от связанной функциональности (Leffingwell, 2011). Команды гибкой разработки измеряют свою *скорость*, число баллов историй, которые команда реализует в стандартной итерации и которые оцениваются на основе предыдущего опыта и результатов ранних итераций нового проекта. При оценке графика, затрат и числа необходимых итераций проекта члены команды учитывают размер резерва продукта (product backlog) и свою скорость разработки. Дин Леффингуэлл (Dean Leffingwell, 2011) описывает несколько таких приемов оценки и планирования проектов гибкой разработки.

> **Внимание!** Если вы не сравните свои расчеты с фактическими результатами проекта и не улучшите свои навыки расчета, то ваша оценка вечно будет оставаться всего лишь предположением. Чтобы собрать достаточно данных для корреляции размера ПО с усилиями на его разработку, потребуется время. Первые итерации проектов гибкой разработки позволяют команде оценить скорость.

Даже самые лучшие методики не сработают, если клиенты, менеджеры или юристы будут часто менять требования. Если изменения настолько велики, что разработчики не успевают их учитывать, команда может впасть в ступор, не имея возможности продолжать проект. Методы гибкой разработки — еще один способ решения проблемы высокой изменчивости требований. При использовании этих методов сначала реализуется сравнительно небольшая часть требований, при этом заранее известно, что в будущем будут вноситься изменения. После этого на ранних итерациях команда уточняет оставшиеся требования к продукту на основе отзывов клиентов.

Цель не одно и то же, что оценка. Несоответствие определенного срока и тщательно продуманного графика разрешается в процессе обсуждения. Менеджер проекта, способный обосновать оценку на основе тщательно продуманных процессов и исторических данных, лучше подготовлен к обсуждению, чем тот, что делает оценки из «общих соображений». Бизнес-цели проекта должны служит основанием, на котором заинтересованные лица разрешают конфликты за счет продления графика, сокращения границ, добавления ресурсов или снижения качества. Это непростые решения, но это единственный путь максимизации ценности создаваемого продукта.

Есть часок?

Как-то клиент попросил наших разработчиков адаптировать небольшую программу, которую он написал для себя, к нашей общей компьютерной сети, чтобы и его коллеги могли ее использовать. «Есть час?» — спросил меня наш менеджер, передавая мне поверхностную оценку размера проекта. Когда я порасспросил клиента и его коллег, чтобы понять, что же им требуется, оказалось, что проблем несколько больше. Я потратил 100 часов только на написание программы, которая им была нужна, без наведения лоска. Тот факт, я потратил в 100 раз больше времени, чем рассчитывал менеджер, говорит о том, что его расчеты были несколько поспешными и основывались на неполной информации. В команде должны изучить требования, оценить масштаб и составить представление о размере до того, как кто-нибудь приступит к расчетам или возьмет на себя определенные обязательства.

Неясно сформулированные требования неизбежно приводят к неточным расчетам. Поскольку определенные неясности в требованиях на ранних стадиях проекта неизбежны, заложите в график и бюджет резерв для непредвиденных обстоятельств, чтобы учесть некоторое разрастание требований (Wiegers, 2007). Оно происходит из-за изменения ситуации в бизнесе, сме-

щения приоритетов пользователей и рынка, а также того, что со временем заинтересованные в проекте лица лучше начинают понимать, что же продукт может и должен делать. В проектах гибкой разработки расширение границ обычно вызывает добавление дополнительных итераций в цикл разработки. Однако, значительное увеличение требований, как правило, указывает на то, что многие требования были упущены в ходе сбора информации.

> **Внимание!** Не позволяйте, чтобы на ваши расчеты влияли желания других. Ваши прогнозы не должны меняться только потому, что кому-то они не нравятся. Слишком большое несоответствие в прогнозах указывает на необходимость переговоров.

Требования и график

Во многих проектах применяется «планирование справа налево»: сначала устанавливают жесткую дату поставки продукта, а затем определяют требования. В таком случае разработчики зачастую часто не успевают закончить работу в поставленные сроки, включив в ПО всю необходимую функциональность соответствующего уровня. Более реалистично определить требования к ПО *до* составления подробных планов и принятия определенных обязательств. Тем не менее стратегия «дизайн по расписанию» может сработать, если менеджеру проекта удастся договориться с заказчиком, какая часть желаемой функциональности удовлетворит его в указанные сроки. Как и всегда, расстановка приоритетов требований — ключевой фактор для успеха проекта.

Для сложных систем, в которых ПО является лишь частью конечного продукта, детальные графики, как правило, составляются после разработки требований на уровне продукта (системы) и предварительного определения архитектуры. На этом этапе ключевые даты поставки могут быть определены и согласованы на основе информации из таких источников, как отдел маркетинга, отдел продаж, отдел по работе с клиентами и разработчики.

Подумайте о поэтапном планировании и финансировании проекта. На стадии первоначального изучения требований вы получите достаточно информации, чтобы составить реалистичные планы и расчеты для одного или более этапов разработки. Проекты, требования к которым сформулированы нечетко, выиграют в случае применения поэтапного цикла разработки. Эта модель позволит команде приступить к созданию качественного ПО задолго до полного выяснения требований. Расставив приоритеты требований, вы сможете определить очередность включения функциональности на каждом этапе.

Часто причина неудачи проекта по разработке ПО кроется в том, что разработчики и другие участники проекта слишком оптимистичны и плохо составляют планы, а не в том, что они плохие специалисты. К главным ошибкам планирования относятся пропуск стандартных задач, недооценка затрат или сроков, а также проектных рисков и упущение возможности переделок. Кроме того, вы можете забыть учесть возможные переделки (McConnell, 2006). Эффективное планирование проекта предполагает наличие следующих элементов:

- оценку размера продукта;
- информацию о производительности специалистов, основанной на прошлом опыте;
- список задач, которые необходимы для полной реализации или проверки функции или варианта использования;
- требования с приемлемым уровнем стабильности, по крайней мере для будущей итерации разработки;
- опыт, который позволит менеджеру проекта учесть нематериальные факторы и индивидуальные особенности каждого проекта.

> **Внимание!** Не поддавайтесь давлению и не принимайте на себя невыполнимых обязательств. Это верный путь к неудаче.

От требований — к дизайну и коду

Граница между требованиями и дизайном не четкая, а серая, размытая область (Wiegers, 2006). Постарайтесь не делать в ваших спецификациях упор на реализацию, кроме тех случаев, когда имеется веская причина для намеренного ограничения дизайна. В идеале, на описание предполагаемых функций системы не должно влиять представление о дизайне. Надо признать, что во многих проектах учитываются ограничения дизайна, налагаемые разработанными ранее продуктами, стандартами семейства продуктов и соглашениями о пользовательском интерфейсе. Именно поэтому в спецификации к требованиям практически всегда содержится некоторая информация о дизайне. Постарайтесь избежать случайных элементов дизайна, ненужных или незапланированных ограничений конечного дизайна. Подключите дизайнеров к экспертизе требований, чтобы удостовериться, что требования могут служить надежной основой для дизайна.

Архитектура и выделение ресурсов

Требования к продукту, атрибуты качества и характеристики пользователей влияют на архитектуру продукта (Bass, Clements и Kazman, 1998; Rozanski и Woods, 2005). Изучение предлагаемой архитектуры помогает аналитикам проверить требования и отшлифовать их точность, как и при использовании прототипов. Оба метода базируются на следующем посыле: «Если я правильно понимаю требования, то способ, который я сейчас проверяю, позволит удовлетворить их. Теперь же, когда у меня в руках предварительная архитектура (или прототип), поможет ли это мне лучше разобраться в требованиях и обнаружить неправильные, отсутствующие или противоречивые требования?»

Архитектура особенно важна для систем, в которые входят компоненты ПО и оборудования, а также для сложных систем, состоящих исключительно из ПО. Важный этап анализа требований — распределение высокоуровневых требований к системе по различным подсистемам и компонентам. Системный

инженер, аналитик или архитектор классифицирует требования к системе по их принадлежность к функциональным требованиям и требованиям к оборудованию. Информация о связях требований позволяет разработчикам отследить, в каких элементах дизайна будет отражено то или иное требование.

В результате неверных решений по назначению требований может статься, что ПО, определенное как функциональное, следовало бы назначить компонентам оборудования (или наоборот); кроме того, в таком случае возможна низкая производительность или трудности при замене некоего компонента для улучшения версии. При разработке одного проекта инженер по оборудованию сообщил моей группе, что в нашем ПО должны быть преодолены все ограничения, налагаемые его дизайном оборудования! Хотя ПО само по себе более гибко, чем оборудование, инженерам не следует использовать это качество, чтобы экономить на дизайне оборудования. Воспользуйтесь инженерным подходом, чтобы решить, какие возможности каждый системный компонент будет удовлетворять.

Распределение системных функций по подсистемам и компонентам надо выполнять сверху вниз (Hooks и Farry, 2001). Представьте проигрыватель дисков Blu-Ray, в который входит мотор, открывающий и закрывающий лоток дисковода и вращающий диск, оптическая подсистема для считывания данных с диска, ПО для формирования изображений, многофункциональный пульт дистанционного управления и многое другое (рис. 19-3). Подсистемы взаимодействуют, определяя управление поведением в таких случаях, когда пользователь нажимает кнопку на пульте дистанционного управления, чтобы открыть лоток дисковода во время воспроизведения диска. В таких сложных продуктах требования к системе влияют на дизайн архитектуры, а архитектура в свою очередь влияет на распределение требований.

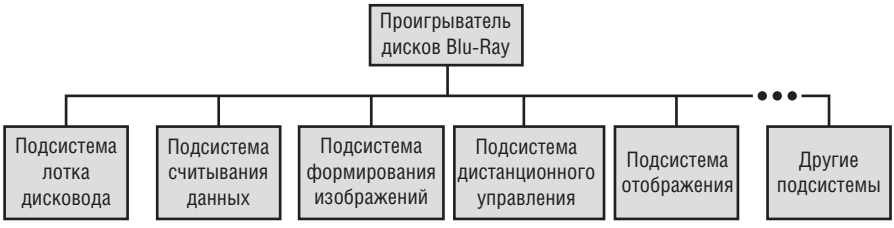


Рис. 19-3. Такие сложные продукты, как проигрыватели дисков Blu-Ray, содержат много подсистем ПО и оборудования

Невероятно уменьшившийся проект

Как-то я занимался проектом, имитирующим поведение фотографической системы с помощью восьми вычислительных процессов. После скрупулезного анализа требований, команде не терпелось приступить к написанию кода. Но вместо этого нам пришлось создавать модель — в тот момент мы больше думали о создании решения, а не пытались разобраться в проблеме. Мы быстро поняли, что в трех этапах моделирования использова-

лись идентичные вычислительные алгоритмы, в трех других применялся еще один набор алгоритмов, а в остальных двух — третий набор. Взгляд с позиций дизайна позволил упростить проблему и уменьшить число сложных вычислений с восьми до трех. Начни мы кодировать сразу же после анализа требований, в какой-то момент мы бы без сомнения обнаружили повторение кода, но мы сэкономили массу времени, выявив их гораздо раньше. Пересмотр моделей дизайна намного эффективнее, чем переделка кода!

Дизайн ПО

При разработке некоторых проектов дизайн ПО немного изменяется, тем не менее на дизайн время никогда не бывает потрачено зря. Разнообразные разработки ПО удовлетворяют большинству требований к продукту. Они различаются по производительности, эффективности и устойчивости к сбоям, а также по использованным техническим методам. Если вы перескакиваете от требований к коду, это означает, что вы по существу разрабатываете ПО «на лету». Вы разрабатываете какой-то дизайн, но не обязательно идеальный, вероятный результат в этом случае — плохо структурированное ПО.

Как и в случае с требованиями, отличный дизайн — это результат итераций. Многократно прорабатывайте дизайн для уточнения первоначальной концепции по мере получения информации и появления дополнительных идей. При недостатках в дизайне появляются продукты, которые трудно обслуживать и расширять, которые не удовлетворяют пожеланиям клиентов в отношении производительности, легкости и простоты использования и надежности. Время, затраченное на преобразование требований в дизайн, — это отличная инвестиция в создание высококачественных и надежных продуктов.

Проект, в котором применяются методы объектно-ориентированной разработки, может начаться с объектно-ориентированного анализа требований с применением диаграмм классов и других UML-моделей для представления и анализа информации требований. Дизайнер может переработать эти концептуальные диаграммы классов, не зависящие от особенностей реализации, в более подробные диаграммы классов для объектно-ориентированного дизайна и реализации.

Вам не нужно разрабатывать полный подробный дизайн всего продукта до начала реализации, однако следует выполнить дизайн каждого компонента до того, как вы приступите к его реализации. Планирование дизайна наиболее полезно для особенно трудных проектов, где разрабатываются системы с множеством внутренних граничащих и взаимодействующих друг с другом компонентов, а также тех, которыми занимаются неопытные разработчики (McConnell, 1998). Однако проекты любого типа выиграют, если вы:

- разработаете надежную архитектуру подсистем и компонентов, которая обеспечит поддержку будущих изменений;

- определите ключевые классы объектов или функциональные модули, которые необходимо создать, определив их интерфейсы, функции и взаимодействие с другими элементами;
- убедитесь, что в дизайн включены все функциональные требования и что он не содержит ненужной функциональности;
- определите предполагаемую функциональность каждой единицы кода, которая следует основному принципу дизайна: сильная связанность, слабая связь и сокрытие информации (McConnell, 2004);
- удостоверитесь, что дизайн учитывает все условия исключения, которые могут возникнуть;
- убедитесь, что дизайн будет отвечать заданным критериям производительности, безопасности и другим целям по качеству;
- убедитесь, что существующие компоненты можно использовать повторно;
- определите и соблюдайте все ограничения, оказывающие значительное влияние на дизайн программных компонентов.

По мере преобразования требований в дизайн и код, разработчики сталкиваются с определенными неясными и запутанными местами. В идеале они могут обратиться к клиентам или бизнес-аналитиками за решением, используя применяемый в проекте процесс отслеживания дефектов. Если же проблему нельзя решить немедленно, любые предположения, догадки или интерпретации разработчики должны задокументировать и отправить на просмотр представителям клиента.

Дизайн пользовательского интерфейса

Дизайн пользовательского интерфейса — активно изучаемая предметная область, которая выходит далеко за пределы обсуждаемых в этой книге тем. В наших усилиях по выявлению требований мы лишь вскользь коснулись дизайна пользовательского интерфейсе. Он настолько тесно связан с требованиями, что не следуют быстро передавать требования в разработку, не скоординировав дизайн с конечными пользователями. В главе 15 рассказывается, как варианты использования ведут к картам диалоговых окон. Для документирования элементов пользовательского интерфейса и его реакции на действия пользователей очень полезной оказывается модель «отображение-действие-реакция» (display-action-response, DAR). Модель DAR объединяет визуальный макет экрана с таблицами, описывающими его элементы и их поведение в различных условиях.

На рис. 19-4 показан пример страницы веб-сайта, а на рис. 19-5 — соответствующая ей модель «отображение-действие-реакция». Эта модель содержит достаточно подробностей о макете и поведении страницы, чтобы разработчик мог уверенно приступить к ее реализации.

Рис. 19-4. Высококачественный дизайн веб-страницы

Элемент пользовательского интерфейса: страница Submit a Pearl (Добавить жемчужину) сайта PearlsFromSand.com	
Описание элемента пользовательского интерфейса	
Идентификатор	submit.html
Описание	Страница, на которой пользователи могут добавить свои «истории из жизни» для размещения в блоге сайта Pearls from Sand
Предварительное условие	**Экран**
Всегда	Ссылка Home (Основная) Ссылка About the Book (О книге) Ссылка About the Author (Об авторе) Ссылка Blog (Блог) Ссылка Submit a Pearl (Добавить жемчужину) (неактивна и выделена другим цветом, потому что это текущая страница) Ссылка Buy the Book (Купить книгу) Ссылка Contact (Контакты) Текстовое поле Name (Имя) Текстовое поле City (Город) Поле со списком State or Province (Штат или провинция) Текстовое поле Email (Электронная почта) Текстовое поле Title (Заголовок) Поле со списком Pearl Category (Категория жемчужины) Текстовое поле Your Story (Текст вашей истории) Флажок I agree (Согласен), в снятом состоянии Кнопка Submit (Отправить) Ссылка Pearl Submission Guidelines (Советы по размещению жемчужин) Ссылка Pearl Submission Terms (Условия размещения жемчужин)

Пользователь только что разместил жемчужину	Поля Name, City, State or Province и Email заполняются значениями из предыдущей жемчужины. Поля Title, Pearl Category, Your Story и I agree заполняются значениями по умолчанию

Поведение элементов пользовательского интерфейса

Предварительное условие	Действие пользователя	Реакция
Всегда	Пользователь щелкает ссылку: Home, About the Book, About the Author, Buy the Book, Contact, Pearl Submission Guidelines, Pearl или Submission Terms	Открывается соответствующая страница
Всегда	Пользователь щелкает любую из ссылок Blog	Блог Pearls from Sand открывается в новой вкладке браузера
Всегда	Пользователь вводить или вставляет текст в текстовое поле	Текст пользователя отображается в поле Your Story, также отображается число оставшихся символов
Всегда	Пользователь щелкает флажок I agree	Состояние флажка изменяется по циклу «включен-отключен»
Одно или несколько неверных значений в элементах управления	Пользователь щелкает ссылку Submit	Отображается сообщение об ошибке, о неверном тексте, длине или незаполненных обязательных полях
Все поля содержат верные значения, флажок I agree установлен	Пользователь щелкает ссылку Submit	Жемчужина принимается, счетчик жемчужин увеличивается, сообщение электронной почты с жемчужиной отправляется пользователю и администратору, отображается сообщение с подтверждением принятия жемчужины
Флажок I agree не установлен	Пользователь щелкает ссылку Submit	Система отображает сообщение об ошибке на текущей странице

Рис. 19-5. Модель «отображение-действие-реакция» веб-страницы на рис. 19-4

От требований — к тестированию

Тестирование и разработка требований тесно связаны. Согласно мнению консультанта Дороти Грехем: «Чем лучше требования, тем качественнее тесты; чем качественнее анализ тестирования, тем лучше требования» (Graham,

2002). Требования обеспечивают основу для тестирования системы. Продукт следует тестировать на соответствие тому, что он, как записано в требованиях, должен делать, а не на предмет дизайна или кода. Тестирование системы, выполненное на основе кода, может стать самосбывающимся предсказанием. Продукт может с вести себя именно так, как описано в вариантах тестировании, созданных на основе кода, но это не означает, что он соответствует потребностями пользователей. Подключите тестировщиков к рецензированию требований, чтобы убедиться, что требования поддаются проверке и могут служить основой для тестирования системы.

В командах гибкой разработки обычно вместо точных требований пишут приемочные тесты (Cohn, 2004). Вместо определения качеств, которые должна демонстрировать система или действий, которые должен иметь возможность выполнять пользователь, приемочные тесты формулируют ожидаемое поведение пользовательской истории. Это дает разработчикам информацию, которая позволяет им быть уверенными, что они правильно и полно реализовали все истории. Как говорится в главе 17, приемочные тесты должны охватывать:

- ожидаемое поведение в нормальных условиях (хорошие входные данные и корректные действия пользователя);
- обработку ожидаемых условий ошибок и ожидаемых сценариев сбоев (плохие входные данные и некорректные действия пользователя)
- выполнение ожиданий по качеству (например, время оклика, средства безопасности и среднее время или число выполнения задачи пользователем).

Что тестировать?

Участник семинара как-то заметил: «Я работаю в группе тестирования. У нас нет написанных требований, поэтому мы должны протестировать систему на предмет того, что она с нашей точки зрения должна делать. Иногда мы ошибаемся, тогда мы узнаем у разработчиков, каковы функции системы, и тестируем ее снова».

Тестирование того, что разработчики создали, — не то же самое, что тестирование того, что они должны были создать. Требования — это конечный документ для системных и пользовательских приемочных испытаний. Если требования к системе сформулированы плохо, то тестировщики обнаружат множество требований, самостоятельно выведенных — верно или неверно — и реализованных разработчиками. Аналитик должен задокументировать правомерно подразумеваемые требования и их источники, чтобы будущее регрессионное тестирование было более эффективным.

Тестировщики проекта должны определить, как они будут проверять каждое требование. Возможных методов несколько:

- тестирование (выполнение ПО с целью поиска дефектов);

- экспертиза (проверка кода на предмет его соответствия требованиям);
- демонстрация (демонстрация того, что продукт работает, как ожидалось);
- анализ (обоснование того, как система должна работать при определенных условиях).

Увязка тестирования с требованиями помогает выдерживать приоритеты тестирования и фокусироваться на получении максимальной пользы. Одна коллега, опытный менеджер проектов и бизнес-аналитик, изложила свой опыт так: «Приемочное тестирование пользователями может обусловливаться четко сформулированной бизнес-потребностью и обычно является последним препятствием, проходимым проектом перед выпуском продукта. В недавнем проекте разработки веб-портала мы работали с куратором над пониманием того, какие преимущества должен обеспечить веб-портал. Понимание критически важных требований позволило менеджеру проекта сформулировать четкие определения критически важных, умеренно важных и косметических дефектов. За счет увязки критериев дефектов с требованиями мы смогли провести наших клиентов через приемочное тестирование пользователями и успешно завершили основную часть разработки без каких-либо сомнений в критериях качества или приемки».

Само по себе обдумывание того, как проверить каждое требование, является ценным приемом работы над качеством. Используйте такие приемы анализа, как графики типа «причина-результат», чтобы составить варианты тестирования на основе логики, описанной в требовании. При этом будут выяснены неясности, пропуски или *другие* неявные предположения и другие проблемы. Необходимо, чтобы каждое функциональное требование можно было проследить по крайней мере до одного варианта тестирования в системном тестовом наборе, для того чтобы ни одна ожидаемая реакция системы не осталась непроверенной. В основанном на требованиях тестировании применяются несколько стратегий проектирования тестов: на основе действий, данных (включая анализ граничных значений и эквивалентное разбиение классов), логики, событий и состояния (Poston, 1996). Опытные тестировщики могут усовершенствовать тестирование, основанное на требованиях, за счет использования дополнительных тестов, основанных на истории продукта, предполагаемых сценариях использования, общих характеристиках качества и случайностях.

Усилия, затраченные на обдумывание тестов заранее, не пропадают зря, даже если планируется отдельное тестирование системы перед выпуском. Это вопрос перераспределения усилий по тестированию, которое как правило откладывается на более поздние стадии проекта. Концептуальные тесты легко преобразуются в конкретные сценарии тестирования и поддаются автоматизации, где это возможно и уместно. Переход к тестированию на более раннем этапе цикла разработки окупиться повышением качества требований, более четким взаимодействием и общим пониманием среди заинтересованных лиц, а также ранним устранением дефектов.

По мере продвижения разработки команда будет конкретизировать требования от высоких уровней абстракции, как, например, в вариантах использования, через функциональные требования к ПО к спецификациям для отдельных модулей кода. Авторитетный специалист в области тестирования Борис Бейцер (Beizer, 1999) подчеркивает, что тестирование на основе требований должно выполняться на каждом уровне конструирования ПО, а не только на уровне конечных пользователей. Часть кода в приложении не доступна пользователям напрямую, однако он необходим для внутренних взаимодействий. Каждый модуль должен удовлетворять собственной спецификации, даже если функция этого модуля невидима для пользователя. Вследствие этого тестирование системы на основе пользовательских требований — необходимая, но не достаточная стратегия тестирования системы.

От требований — к успеху

Недавно я наблюдал такую ситуацию: разработчики-контрактники присоединились к проекту для реализации приложения, требования для которого писала другая команда. Новички только взглянули на десяток пухлых переплетов документации по требованиям, вздрогнули от ужаса и начали программировать. В ходе сборки они не обращались к спецификации. Вместо этого, они построили то, что, по их мнению, задумывалось, основываясь на неполном и неточном понимании задач будущей системы. Неудивительно, что возникло множество проблем. Перспектива разбираться в массе даже самых совершенных требований без сомнения приводит в уныние, но игнорирование требований — это верный путь к провалу проекта.

Гораздо быстрее прочитать требования, какими бы объемными они не были, до реализации, чем создавать ненужную систему, а затем переделывать ее. Еще лучше привлечь разработчиков в начале проекта, чтобы они смогли поучаствовать в работе над требованиями, в создании прототипов или воспользоваться итеративным подходом к разработке. Но разработчики рано или поздно должны прочитать всю спецификацию. Чтение растягивается на весь проект, что немного облегчает тяжесть этой работы.

Практика более успешного проекта такова: составляется список всех требований, которые были включены в определенную версию. Группа контроля качества проекта оценивает каждую версию, тестируя все требования. То из них, условия которого не удовлетворяло критериям тестирования, считалось дефектом. Группа контроля качества откладывала выпуск версии, если не были удовлетворено количество требований, превышающее заранее оговоренное, или если крайне важные требования. В основе успеха проекта лежало использование документированных требований для принятия решения о готовности версии к выпуску.

Конечным, подлежащим сдаче продуктом считается программная система, отвечающая потребностям и ожиданиям клиента. Требования являются важным этапом на пути от бизнес-потребностей до удовлетворенных клиен-

тов. Если в основе ваших планов проекта, дизайна ПО и системного тестирования не лежат высококачественные требования, вероятнее всего, вам придется затратить много усилий на создание качественного продукта. Однако не становитесь и рабом требований. Не имеет смысла тратить массу времени на создание ненужной документации и проведение ритуальных встреч, если при этом не создается ПО и проект закрывается. Попытайтесь достичь разумного баланса между доскональной спецификацией и кодированием «из головы» — это снизит до приемлемого уровня риск создания ненадлежащего продукта.

Что дальше?

- Оцените объем работы над требованиями в вашем следующем проекте с помощью средства оценки требований с рис. 19-2. Засеките время своей работы над проектом и сравните результаты со своей начальной оценкой. Примените средство оценки к своему следующему проекту.
- Рассчитайте средний процент незапланированного роста проекта для нескольких последних проектов. Можете ли вы включить резерв для непредвиденных обстоятельств в график проекта для учета такого увеличения объема в будущих проекта? Используйте данные об увеличении объема из предыдущих проектов для объяснения этого резерва, чтобы он не выглядел как произвольное решение.
- Попробуйте отследить все требования в реализованной части спецификации требований к ПО до отдельных элементов дизайна. Это можно отразить в моделях потока данных, таблице в диаграмме «сущность-связь», объектных классах или методах или других элементах дизайна. Нет ли отсутствующих элементов дизайна? Или, может, были пропущены какие-то требования?
- Запишите количество строк кода, функциональных точек, классов объектов или элементов пользовательского интерфейса, которые необходимы для реализации каждой функции продукта или варианта использования. Укажите вашу оценку затрат, необходимых для полной реализации и проверки каждой функции или варианта использования, а также реальные затраты. Выявите взаимосвязи размера продукта и затраченных усилий — это поможет вам выполнить более точные расчеты для будущих спецификаций к требованиям.
- Запишите свои оценки размера и объема усилий на разработку требований и готовых компонентов в своем проекте и сравните их с фактическими результатами. Вам действительно хватило пяти интервью или пришлось все-таки провести 15? Создали ли вы вдвое больше вариантов использования, чем ожидалось? Как нужно изменить процесс, чтобы получать более точные оценки в будущем?

Часть III

Требования в проектах определенных классов

Глава 20

Проекты гибкой разработки (agile)

Под *гибкой разработкой* (agile development) подразумевают набор методов разработки ПО, обеспечивающих постоянное сотрудничество между заинтересованными лицами и быстрый и частый выпуск функциональности небольшими порциями. Существует много разных типов гибкой разработки, вот лишь самые популярные: Scrum, Extreme Programming, Lean Software Development, Feature-Driven Development и Kanban. Термин «гибкая разработка» приобрел популярность с момента публикации «Manifesto for Agile Software Development» («Манифест гибкой разработки ПО») (Beck и др., 2001). Методы гибкой разработки основаны на интерактивных и инкрементальных подходах к разработке ПО, которые существуют уже много лет (см. например, Boehm, 1988; Gilb, 1988 и Larman и Basili, 2003).

Методики гибкой разработки обладают разными характеристиками, но в основном в них предпочтение отдается адаптивному (иногда он называется «управляемым изменениями»), а не прогнозному (иногда он называется «плановым») подходу (Boehm и Turner, 2004; IIBA, 2009). В прогнозном подходе, таком как водопадная методика, пытаются минимизировать риски в проекте, выполняя значительный объем планирования и документирования до начала создания ПО. Менеджеры проектов и бизнес-аналитики обеспечивают, чтобы все заинтересованные лица четко понимали, что планируется выпустить, до начала разработки. Это работает, если с самого начала есть четкое понимание требований, а сами требования остаются практически неизменными на протяжении всего проекта. Адаптивные подходы, такие как методы гибкой разработки, призваны обеспечить приспособляемость к неизбежным изменениям, которые будут происходить в проектах. Они также хорошо работают в проектах с неточными или изменчивыми требованиями.

В этой главе описываются особенности методик гибкой разработки в применении к разработке требований, основные изменения традиционных приемов работы с требованиями, а также «дорожная карта», где найти более подробные сведения по теме в данной книге.

Гибкие требования?

Мы не используем термин «гибкие требования», потому что это подразумевало бы, что требования в проекте гибкой разработки чем-то отличаются от проектов других жизненных циклов. Разработчикам нужен одинаковый объем информации, чтобы правильно реализовывать нужную функциональность во всех проектах. Однако в обычных проектах и проектах гибкой разработки с требованиями работают по-разному, особенно в том, что касается глубины проработки, места требований в графике проекта, а также объема письменной документации требований. Вот почему мы используем термин «требования в проекте гибкой разработки».

Ограничения водопадной модели

Часто считают, что водопадный процесс разработки предусматривает линейную последовательность операций, в которой в команде проекта определяют (иногда излишне) требования, после чего создают дизайн, затем пишут код и, наконец, тестируют решение. Теоретически, у этого подхода есть ряд преимуществ. В команде могут обнаружить все недостатки требований к приложению и дизайну на ранних этапах, а не на стадиях разработки, тестирования или сопровождения, когда исправление ошибок обходится значительно дороже. Если требования верны с самого начала, легко определять бюджет и ресурсы, оценивать прогресс и точно определять дату завершения. Но в реальности разработка ПО редко бывает столь прямолинейной.

В очень немногих проектах применяется чистый последовательный водопадный подход. Даже в прогнозируемых проектах ожидается определенный объем изменений, и нужны механизмы для работы с ними. Всегда есть некоторое перекрытие и обратная связь между стадиями. Но в целом, в проектах водопадной разработки команда прикладывает значительные усилия к тому, чтобы как можно раньше «как положено» создать все требования. Помимо водопадной модели и модели гибкой разработки существует много других жизненных циклов разработки ПО. В них по-разному относятся к разработке полного набора требований на ранних этапах проекта (McConnell, 1996; Boehm и Turner, 2004). Основной отличительный признак всего спектра проектов — от фиксированных, прогнозируемых проектов и полностью неопределенных, адаптивных проектов — заключается во времени, которое проходит от создания требования до поставки основанного на этом требовании ПО клиенту.

В крупных водопадных проектах поставка продукта часто происходит с задержкой, в нем отсутствуют необходимые функции, а сам продукт не оправдывает пользовательских ожиданий. Водопадные проекты подвержены таким неудачам из-за многих слоев зависимостей в требованиях. В процессе длинного проекта заинтересованные лица часто меняют свои требования, процесс разработки стопорится из-за неспособности разработчиков эффективно реагировать на эти изменения. Реальность такова, что заинтересованные лица

будут менять требования — потому что в начале проекта они не знают точно, что им нужно, потому что способны сформулировать свое видение, только увидев что-то, что точно не соответствует их видению и потому что иногда в процессе выполнения проекта их бизнес-потребности меняются.

Первую публикацию формальной водопадной модели (хотя и под другим именем) приписывают Уинстону Ройсу (Royce, 1970), но он представил ее в контексте описания подхода, который «рискован и чреват неудачами». Он точно определил проблему, с которой сегодня сталкиваются при реализации проектов: ошибки требований не обнаруживаются до начала тестирования, то есть до поздних стадий проекта. Далее Уинстон объяснил, что этапы проект *в идеале* должны выполняться в последовательности: требования, дизайн, кодирование и тестирование, но также он заметил, что в проектах *действительно* требуется перекрытие некоторых из этих фаз и итерации между ними. Он даже предложил в качестве эксперимента выполнять моделирование для создания прототипов требований и дизайна до начала полноценной разработки. Вместе с тем видоизмененная водопадная модель сегодня применяется во многих проектах, с переменным успехом.

Разрушительные изменения бизнес-целей

По истечении года работы водопадного проекта новый директор по маркетингу взял на себя роль куратора проекта. В команде уже разработали большой объем ПО, но еще ничего полезного для клиентов не развертывалось. Неудивительно, что бизнес-цели нового куратора отличались от целей предшественника. Бизнес-аналитик поделился с разработчиками новостями о том, что появились новые бизнес-цели, а, значит, новые пользовательские и функциональные требования, а еще пересмотрены приоритеты старых требований.

Разработчики привыкли переносить все новые требования в пакет обновления, который планировалось выпустить через какое-то время после начального развертывания. Они взбунтовались, протестуя против недопустимого изменения курса в самом разгаре проекта. Однако новый куратор считал неприемлемым продолжение разработки и поставку продукта, удовлетворяющего только начальным требованиям. Если бы в команде применяли подход, предусматривающий изменения и адаптацию к ним, то подобное изменение стратегического направления не было бы столь разрушительным.

Гибкий подход к разработке

В методах гибкой разработки есть определенные попытки преодолеть ограничения водопадной модели. Методы гибкой разработки ориентированы на итеративный и инкрементальный процесс программирования, в котором разработка ПО разбивается на короткие циклы, называемые итерациями (в методе гибкой разработки под названием Scrum они называются «спринтами»).

Продолжительность итераций варьируется от одной недели до месяца. На протяжении итерации команда разработчиков добавляет небольшой набор функциональность, основываясь на определенных клиентом приоритетах, тестирует на предмет корректной работы и проверяет, руководствуясь определенными клиентом критериями приемки. В последующих итерациях меняется уже сделанное, расширяются исходные функции и устраняются обнаруженные дефекты. Постоянное участие клиента позволяет команде обнаруживать проблемы и отклонения от правильного направления, что дает разработчикам вовремя вносить коррективы, пока дело не зашло слишком далеко. Задача заключается в получении готового к поставке ПО в конце итерации, даже если это всего лишь небольшая часть требуемого конечного продукта.

Особенность гибкой разработки в применении к требованиям

В следующих разделах рассказывается об отличиях проектов гибкой разработки и традиционных проектов в том, что касается требований. Многие приемы работы с требованиями в проектах гибкой разработки также хорошо работают — и стоит подумать об их применении — в других жизненных циклах разработки.

Вовлечение клиентов

Сотрудничество с клиентами в процессе разработки ПО всегда повышает шансы проекта на успех. Это верно как в водопадных проектах, так и проектах гибкой разработки. Основное различие между этими методиками заключается во времени вовлечения клиентов. В водопадных проектах клиенты обычно с самого начала выделяют значительное время на помощь бизнес-аналитикам в понимании, документировании и проверке требований. Клиенты должны также участвовать в проекте на этапах приемочного тестирования пользователями, предоставляя информацию о соответствии продукта их потребностям. Однако на стадии разработки клиенты задействованы мало, что затрудняет корректировку проекта при изменении потребностей клиентов.

В проектах гибкой разработки клиенты (или представляющий их владелец продукта) участвуют на всех стадиях проекта. В некоторых проектах гибкой разработки во время итерации начального планирования клиенты работают с командой проекта над определением и приоритизацией пользовательских историй, которые будут служить начальной «дорожной картой» для разработки продукта. Пользовательские истории обычно менее подробны, чем традиционные функциональные требования, поэтому клиенты должны быть доступны в процессе итераций, чтобы предоставлять свои отклики и уточнения во время дизайна и проектирования. Они также должны тестировать и предоставлять свои замечания к новым разработанным функциям, когда завершается фаза разработки в текущей итерации.

Очень часто владельцы продукта, клиенты и конечные пользователи участвуют в написании пользовательских историй и других требований, но эти люди не всегда обучены эффективной разработке требований. Написанные неквалифицированным участником пользовательские истории скорее предоставляют недостаточно точную информацию о требованиях. Независимо от того, кто пишет пользовательские истории, кто-то с солидным опытом и знаниями бизнес-аналитика должен редактировать их до того, как команда начнет их реализовывать. В главе 6 подробнее рассказывается об участии клиентов в проектах гибкой разработки.

Подробнее о документации

Так как в водопадных проектах на этапе разработки программисты практически не общаются с клиентами, в требованиях должны подробно описываться поведение системы, связи между данными и ожидания в области удобства использования. Тесное сотрудничество клиентов с разработчиками в проектах гибкой разработки обычно означает, что требования могут документироваться менее подробно, чем в традиционных проектах. Бизнес-аналитики и другие ответственные за требования люди могут добиваться необходимой точности в процессе общения и документирования по мере необходимости (IIBA, 2013).

Некоторые считают, что в командах гибкой разработки не нужно писать требования. Это не совсем так — методы гибкой разработки должны способствовать созданию минимального объема документации, которое дает точные указания разработчикам и тестировщикам. Любая документация помимо той, что нужна разработчикам и тестировщикам (или необходима для выполнения требований законов и нормативных документов), представляет собой выполненную впустую работу. В некоторых пользовательских историях может предоставляться немного подробностей, а детально описывается только самая рискованная и важная функциональность, обычно в виде приемочных тестов.

Резерв проекта и расстановка приоритетов

Резерв (backlog) проекта гибкой разработки содержит список запросов на работу, которую должна выполнить команда (IIBA, 2013). Резервы проектов обычно состоят из пользовательских историй, но в некоторых командах их наполняют другими требованиями, бизнес-процессами и дефектами, которые нужно устранить. В каждом проекте должен быть лишь один резерв (Cohn, 2010). Поэтому может потребоваться представлять в резерве дефекты, чтобы можно было определять их приоритеты в одном ряду с новыми пользовательскими историями. В некоторых командах дефекты переопределяют как новые пользовательские истории или варианты старых историй. Резервы могут реализовываться в виде карт историй или в специализированных инструментах. Пуристы от гибкой разработки настаивают на использовании карт, но они неудобны для работы в крупных проектах или в распределенных

командах. Подробнее о резерве продукта см. главу 27. На рынке доступны различные средства управления проектами гибкой разработки, в том числе для управления резервом.

Расстановка приоритетов в резерве выполняется регулярно — одни задачи выбираются на следующие итерации, а другие отбрасываются. Приоритеты, назначаемые элементам резерва, не фиксируются раз и навсегда и могут меняться в следующей итерации (Leffingwell, 2011). Отслеживание связей между элементами резерва и бизнес-требованиями облегчает приоритизацию. Во всех проектах, и не только в тех, где применяется гибкая разработка, нужно управлять приоритетами задач, остающихся в резерве.

Планирование времени

В проектах гибкой разработки применяются те же приемы работы с требованиями, что и в проектах традиционной разработки. Кому-то все равно нужно собирать требования у представителей пользователей, анализировать требования и документировать с различной степенью подробности, а также проверять требования на соответствие бизнес-целям проекта. Однако в проекте гибкой разработки не документируют подробно все требования в самом начале проекта, а первым делом выявляют высокоуровневые требования, обычно в виде пользовательских историй, и заполняют ими резерв проекта, чтобы сразу можно было начать планирование и расстановку приоритетов.

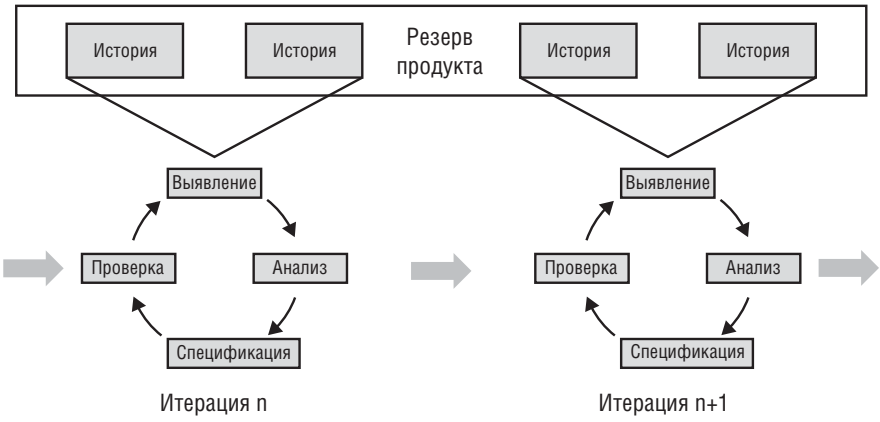


Рис. 20-1. Стандартные операции с требованиями, происходящие на каждой итерации гибкой разработки

Как показано на рис. 20-1, пользовательские истории назначаются для реализации в конкретных итерациях и подробности каждой такой истории дополнительно уточняются в процессе такой итерации. Как показано на рис. 3-3 в главе 3, требования могут разрабатываться небольшими порциями на протяжении всего проекта, и даже вскоре после выпуска проекта. Однако важно как можно раньше определить нефункциональные требования, чтобы можно было спроектировать архитектуру системы, обеспечивающую крити-

чески важные производительность, удобство использования, доступность и другие цели по качеству.

Эпики, пользовательские истории и функции

Как говорилось в главе 8, пользовательская история — это краткое утверждение, которое выражает потребность пользователя и служит исходной точной для общения с целью выяснения деталей. Пользовательские истории специально созданы для обслуживания потребностей «гибких» разработчиков. При изучении пользовательских требований вы вправе задействовать названия вариантов использования, функций или потоков процессов. Выбор формы для описания всех этих требований не важна — в этой главе мы в основном называем их пользовательскими историями, потому что именно они чаще всего используются в проектах гибкой разработки.

Объем пользовательской истории определяют так, чтобы ее реализация могла уложиться в одну итерацию. Майк Кон (Cohn, 2010) определяет *эпику* (epic) как пользовательскую историю, которая слишком велика для реализации в одной итерации. По этой причине эпики нужно разбивать на наборы более мелких историй. Иногда эпики такие большие, что их приходится разбивать на ряд более мелких эпик, каждая их которых в свою очередь разбивается на истории, пока не будут получены истории, которые можно с уверенностью оценить, реализовать и протестировать в одной итерации (см. рис. 20-2). Разбиение больших эпик сначала на более мелкие эпики, а затем на пользовательские истории часто называют *декомпозицией истории* (story decomposition) (IIBA, 2013).

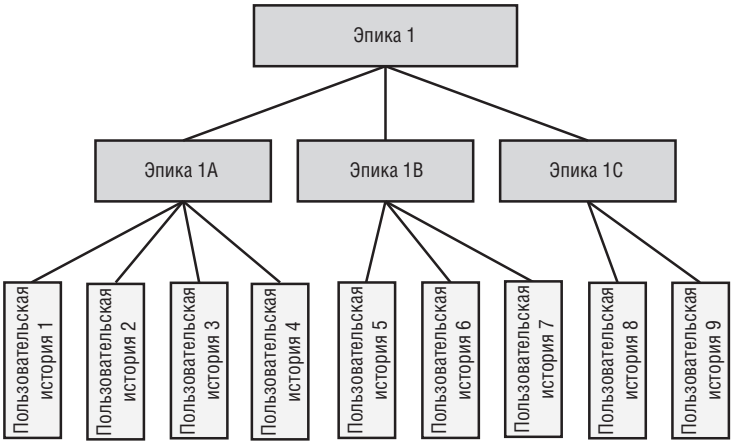


Рис. 20-2. Эпики могут разбиваться на более мелкие эпики, а затем — на пользовательские истории

Функция — это группа возможностей системы, которая представляет ценность для пользователя. В контексте проекта гибкой разработки функции мо-

гут распространяться на одну пользовательскую историю, несколько таких историй, отдельную эпику или даже на несколько эпик. Например, функция увеличения масштаба в фотокамере телефона может разрабатываться для реализации двух не связанных между собой пользовательских историй:

- Будучи матерью, я хочу иметь возможность делать на школьных мероприятиях снимки, на которых четко видна моя дочь, чтобы я могла делиться ими с дедушкой и бабушкой.
- Будучи орнитологом, я хочу иметь возможность на расстоянии получать четкие фотографии птиц, на которых их можно различать.

Определение историй самого нижнего уровня, соответствующих бизнес-требованиям, позволяет определять минимальный набор функциональности, который способна предоставить команда и который одновременно представляет определенную ценность для клиента. Этот принцип иногда называют принципом «минимальной поддающейся для продвижения на рынке функции» (minimal marketable feature, MMF), как описано Марком Денне и Джейн Клеланд-Хуанг (Denne и Cleland-Huang, 2003).

> **Внимание!** При разработке требований в проектах гибкой разработки не стоит уделять много внимания тому, как это называть — историей, эпикой или функцией — важнее сосредоточиться на разработке высококачественных требований, которые позволят разработчикам удовлетворить потребности клиентов.

Расчет на неизбежные изменения

В организациях знают, что в проектах будут изменения. Даже бизнес-цели могут меняться. Самое важное для бизнес-аналитика — перестроиться и при возникновении изменений требования в проекте гибкой разработки говорить не что-то типа: «Минуточку, это выходит за рамки проекта» или «Для внедрения этого изменения нам нужно пройти определенный формальный процесс», а «Хорошо, давайте обсудим изменение». Это способствует привлечению клиента к созданию или изменению пользовательских историй и определению приоритетов всех запросов на изменение по отношению ко всему, что уже есть в резерве проекта. Как и во всех других проектах, в проектах гибкой разработки нужно с умом управлять изменениями, чтобы минимизировать их негативное влияние на проект, но нужно рассчитывать или даже стремиться к неизбежности изменений. Подробнее об управлении требованиями в проектах гибкой разработки см. главу 28.

Понимание своей способности справиться с изменениями не означает, что нужно слепо игнорировать будущее и обращать внимание только на то, что известно в настоящий момент. По-прежнему важно заглядывать в будущее и смотреть, что может вас ожидать далее. Разработчики не обязательно создают дизайн, рассчитанный на все возможные будущие требования. Но имея возможность заглянуть чуть в будущее, они могут создать более расширяемую и надежную архитектуру или закладки в дизайне, облегчающие добавление новой функциональности.

К изменениям относится и удаление элементов из проекта. Удаление может выполняться по самым разным причинам, в числе которых:

- проблемы с реализацией не позволяют завершить элемент в текущий период времени;
- проблемы, обнаруженные владельцами продукта или тестировщиками, делают реализацию той или иной истории неприемлемой;
- необходимость замены менее важных элементов, запланированных на итерацию, более приоритетными.

Адаптация приемов работы с требованиями для проектов гибкой разработки

Большинство описанных в этой книге приемов можно приспособить к проектам гибкой разработки — скорее всего за счет изменения времени и объема их применения, а также лиц, участвующих в их реализации. Международный институт бизнес-анализа (International Institute of Business Analysis, IIBA) предоставляет подробные советы по применению приемов бизнес-анализа в проектах гибкой разработки (IIBA, 2013). Во многих других главах этой книги рассказывается, как описанные в главе приемы адаптировать для применения в проектах гибкой разработки. В табл. 20-1 перечислены главы, в которых непосредственно обсуждаются проекты гибкой разработки.

Табл. 20-1. Перечень глав, в которых обсуждаются приемы гибкой разработки

Глава	Тема
2	Достижение соглашения о требованиях
4	Роль бизнес-аналитика в проектах гибкой разработки, а также кто отвечает за созданные артефакты требований
5	Определение и управление документом о концепции и границах
6	Представительство пользователей
8	Пользовательские истории
10	Определение требований в проектах гибкой разработки
12	Моделирование в проектах гибкой разработки
14	Определение атрибутов качества, в частности необходимых на ранних этапах для создания архитектуры и дизайна
15	Проекты гибкой разработки и эволюционные прототипы
16	Расстановка приоритетов в проектах гибкой разработки
17	Критерии приемки и приемочные тесты
27	Управление требованиями в проектах гибкой разработки с применением резервов и графиков сгорания задач
28	Управление изменениями в проектах гибкой разработки

Переход к гибкой разработке: что дальше?

Если вы бизнес-аналитик, не особо знакомый с методами гибкой разработки, не волнуйтесь: вы по-прежнему можете применять большинство привычных вам приемов. В конце концов и в традиционных проектах и в проектах гибкой разработки командам нужно понимать требования к создаваемым решениям. Вот несколько советов по переходу к методике гибкой разработки:

- Определите свою роль в команде. Как говорится в главе 4, в одних проектах гибкой разработки есть выделенный бизнес-аналитик, а в других задачи бизнес-аналитика выполняют другие люди. Поощряйте ориентацию членов команды на проект, а не на свои индивидуальные роли или должности (Gorman и Gottesdiener, 2011).
- Прочитайте одну из книг о роли владельца продукта в проекте гибкой разработки, чтобы получить хорошее представление о пользовательских историях, приемочных тестах, приоритизации резерва, а также узнать, почему работа бизнес-аналитика продолжается вплоть до конца проекта или выпуска. Можно порекомендовать книгу «Agile Product Management with Scrum» («Гибкое управление продуктом в рамках скрама») (Pichler, 2010).
- Выбрать из предлагаемых приемов те, что лучше всего подойдут вашей организации. Вспомните, какие приемы разработки уже хорошо себя зарекомендовали в вашей организации, и продолжайте их применять. Поговорите с людьми, выполняющими другие роли в команде, и определите, как их приемы работы применить в среде гибкой разработки.
- Сначала реализуйте один небольшой проект в качестве пилотного для обкатки методов гибкой разработки или внедрите несколько приемов гибкой разработки в свой следующий проект.
- Если вы решите реализовать гибридную модель, в которой применяется лишь часть приемов гибкой разработки, для начала выберите только приемы с низким риском, которые хорошо работают в любой методологии. Если вы новичок в гибкой разработке, пригласите опытного наставника на три-четыре итерации, чтобы помочь вам избежать искушения вернуться к привычным приемам работы.
- Не нужно быть пуристом от гибкой разработки только из принципа.

Проявляйте гибкость по отношению к внедрению приемов гибкой разработки

В организации, где я работал, решили перейти от традиционного подхода к гибкой разработке. Вся компания стояла на ушах, догматически пытаясь внедрить приемы гибкой разработки во всей организации сразу. Многие разработчики решили стать пуристами от гибкой разработки, приступив

к написанию карт с историями и ошибочно настаивая, что никакая другая документация не разрешается.

Попытка реализации приемов гибкой разработки с треском провалилась. Не все заинтересованные лица приняли новацию. Некоторые из приемов, на которых настаивали разработчики, невозможно было масштабировать из-за большого размера проектов. Клиенты не понимали, в чем теперь заключается их роль в проекте. Новые проекты провалились с таким треском, что руководство ИТ распорядилось немедленно прекратить переход к гибкой разработке. Все проекты немедленно возвращались к водопадной модели. «Гибкая разработка» стала ругательным словом. Но ведь это похоже на попытку исправить одно неудачное решение другим!

Но сложилась интересная ситуация. Разработчики поняли, что и это распоряжение приведет к проблемам, и поэтому применили смешанный подход к разработке. Они использовали резервы для приоритизации требований, разработка велась трехнедельными итерациями, а также для каждой итерации определялись подробные текущие требования. При рассказе об этом подходе руководству они говорили, что применяют «стандартную водопадную модель» и что проблем быть не должно. На самом деле, большинство приемов гибкой разработки отлично заработали в организации после того, как разработчики научились их правильно применять. Просто в организации изначально неправильно подошли к внедрению приемов гибкой разработки, игнорируя существующие реалии и культуру, — только поэтому гибкая разработка была незаслуженно обругана.

Глава 21

Проекты по доработке или замене систем

В этой книге описана разработка требований для нового ПО или системы, т. е. речь идет о *проекте с чистого листа* (green-field project). Однако многие организации тратят массу сил на обслуживание существующих систем или создание следующих версий уже установленного серийного продукта. Большинство описанных в этой книге приемов подходят для проектов по доработке или замене систем. В этой главе представлены конкретные советы, в каких случаях и как применять те или иные приемы.

Проектом доработки (enhancement project) называются мероприятия по добавлению новых возможностей в существующую систему. Проекты доработки могут также предусматривать устранение дефектов, добавление новых отчетов и модификацию функциональности в соответствии с изменившимися бизнес-правилами или потребностями.

Проект замены (или реинжиниринга) предусматривает замену существующего приложения на новую, разработанную с нуля систему, серийную систему или сочетание таких систем. Проекты замены чаще всего реализовываются для повышения производительности, сокращения затрат (например, на обслуживание или приобретение лицензий), применения современных технологий или для выполнения требований регулирующих органов. Если в проекте замены предусматривается использование серийного решения, стоит почитать подробнее в главе 22.

В проектах замены или доработки есть свои особенности работы с требованиями. Тех разработчиков, которые стояли у истоков системы и держали всю информацию в головах, давно уже нет. Заманчиво заявить, что при добавлении маленького улучшения не стоит писать никаких требований. Разработчики могут посчитать, что не нуждаются в подробных требованиях, если они заменяют функциональность существующей системы. Приемы, описанные в этой главе, помогут вам решать задачи доработки или замены действующей системы, чтобы она лучше удовлетворяла текущие потребности организации.

Случай с недостающей спецификацией

В спецификации требований для следующей версии сформировавшейся системы часто пишут так: «Новая система должна делать то же, что и старая, кроме того, что будут добавлены новые функции и исправлены ошибки». Как-то бизнес-аналитик получила именно такую спецификацию для пятой версии важного продукта. Чтобы точно выяснить, что же текущая версия делает, она заглянула в спецификацию требований к четвертой версии этого ПО. К сожалению, по существу там было сказано вот что: «Версия 4 должна делать то же, что и версия 3, кроме того, что будут добавлены новые функции и исправлены ошибки». Она подняла все предыдущие документы, но так и не смогла найти исходную спецификацию требований. В каждой спецификации перечислялись отличия новой версии от предыдущей, однако нигде не было описания первоначальной системы. Как следствие, у каждого сложилось свое понимание возможностей текущей системы. Если вы оказались в такой ситуации, задокументируйте требования для следующей версии более подробно, чтобы все заинтересованные лица — настоящие и будущие — смогли разобраться в том, что же все-таки система делает.

Ожидаемые затруднения

Присутствие существующей системы создает стандартные проблемы для проектов как доработки, так и замены, в том числе:

- внесенные изменения могут снизить производительность, к которой пользователи привыкли;
- в существующей системе документация по требованиям может быть недостаточной или просто отсутствовать;
- пользователям, знакомым с существующей системой, изменения могут не понравится;
- по незнанию вы можете нарушить или пропустить функциональность, важную для какой-то группы заинтересованных лиц;
- заинтересованные лица могут воспользоваться возможностью запросить новую функциональность, которую, по их мнению, неплохо бы иметь, но которая на самом деле не нужна для достижения бизнес-целей.

Даже если документация есть, она может оказаться бесполезной. В проектах доработки документация может оказаться не соответствующей текущему моменту. Если документация не соответствует текущему состоянию приложения, от нее мало пользы. При замене систем также нужно с осторожностью относиться к переносу *всех* требований, потому может оказаться, что часть старой функциональности не надо переносить.

Одна из самых сложных задач в проектах замены — проверка доводов в пользу замены. Должны быть аргументированные бизнес-цели, оправдывающие замену системы. При полной замене существующих систем могут также

меняться организационные процессы, что усложняет принятие людьми новой системы. Изменения в бизнес-процесах, перемены в системе и необходимость осваивать новую систему могут нарушить текущие процессы.

Работа с требованиями при наличии существующей системы

В табл. 21-1 описаны самые важные приемы разработки требований, которые можно применять при работе над проектами доработки или замены.

Табл. 21-1. Ценные приемы работы с требованиями в проектах доработки или замены

Прием	Цель
Создание дерева функций для представления изменений	Показать добавляемые функции. Выявить функции существующей системы, которых не будет в новой системе
Определение классов пользователей	Оценить, на кого повлияют изменения Выявить новые классы пользователей, потребности которых надо удовлетворить
Определение бизнес-процессов	Понять, как текущая система вплетена в выполнение повседневных задач заинтересованными лицами и как на это повлияют изменения Определить новые бизнес-процессы, которые нужны для новых функций или системы
Документирование бизнес-правил	Зафиксировать бизнес-правила, которые сейчас присутствуют в коде Выявление новых бизнес-правил, которые надо учесть Перепроектировать систему для более качественной обработки изменчивых бизнес-правил, которые был сложны проект поддержке
Создание вариантов использования и пользовательских историй	Понять, что пользователи должны иметь возможность делать в системе Понять, как пользователи представляют себе поведение новых функций Определение приоритетов функциональности для новой системы
Создание контекстной диаграммы	Определить и задокументировать внешние сущности Расширить существующие интерфейсы, добавив в них поддержку новых функций Определить текущие интерфейсы, которые могут потребовать изменений
Создание карты экосистемы	Найти другие системы, на которые оказывается влияние Найти новые, измененные или устаревшие интерфейсы между системами

(*см. след. стр.*)

Табл. 21-1. (*окончание*)

Прием	Цель
Создание карты диалоговых окон	Посмотреть, как новые окна вписываются в существующий пользовательский интерфейс
	Показать, как изменится навигация по окнам процессов
Создание моделей данных	Убедиться, что существующая модель данных достаточна, или расширить ее для поддержки новых функций
	Проверить, что все сущности и атрибуты данных по-прежнему нужны
	Посмотреть, какие данные подлежат миграции, преобразованию, коррекции, архивированию или удалению
Определение атрибутов качества	Убедиться, что проект новой системы удовлетворяет ожиданиям по качеству.
	Улучшить реализацию атрибутов качества по сравнению с существующей системой
Создание таблиц отчетов	Сконвертировать существующие отчеты, которые по-прежнему нужны
	Определить новые отчеты, которых нет в старой системе
Создание прототипов	Привлечь пользователей к процессу создания новой системы
	Создать прототипы основных доработок, если есть неясные места
Изучение спецификаций требований	Найти битые элементы в цепи отслеживания связей
	Определить, нет ли требований, которые устарели и не нужны в новой системе

Проекты доработки предоставляют возможность испробовать новые методы разработки требований безопасно и в малом масштабе. Давление, которое оказывается при выпуске следующей версии, может заставить вас решить, что нет времени на требования. Но проекты доработки позволяют обучаться поэтапно, небольшими шажками. В этом случае, к следующему крупному проекту вы будете чувствовать себя опытным и уверенным, применяя эффективные приемы при разработке требований.

Представьте, что клиент требует добавить новую функцию в уже зрелый продукт. Если вы раньше не имели дела с пользовательскими историями, изучите эту функцию с точки зрения пользовательской истории, обсудив с инициатором запроса задачи, которые клиенты будут выполнять с ее помощью. Попрактиковавшись на этом проекте, вы снизите риск применения пользовательских историй в проекте, разрабатываемом с нуля, в котором только от вашей квалификации может зависеть успех или неудача.

Расстановка приоритетов на основе бизнес-целей

Проекты доработки предпринимают, чтобы добавить новые возможности в существующее приложение. Очень легко увлечься и начать добавлять ненужные функции. Чтобы не добавить в систему ненужных «бантиков», отследите связи требований с бизнес-целями, чтобы убедиться, что новые функции действительно нужны, и выбрать для реализации в первую очередь только самые важные из них. Также нужно определить приоритеты запросов на доработку по отношению к устранению дефектов, обнаруженных в старой системе.

Также очень внимательно следите, чтобы в проекты замены, не прокралась ненужная новая функциональность. Основная задача проектов замены — перенести существующую функциональность. Однако клиенты иногда полагают, что если раз уж разрабатывается новая система, то в нее сразу же можно добавить много новых возможностей. Многие проекты замены рухнули под тяжестью неуправляемого расширения границ проекта. Обычно лучше создать стабильный первый выпуск и добавлять функции в последующих проектах доработки, но при условии, что первый выпуск позволяет пользователям выполнять свою работу.

Проекты замены обычно возникают в ситуации, когда заинтересованные лица хотят добавить функциональность в существующую систему, которая недостаточно гибкая или имеет технологические ограничения. Но должны быть четкие бизнес-цели, оправдывающие реализацию дорогой новой системы (Devine, 2008). Для обоснования проекта замены системы применяйте экономию средств, обеспечиваемую новой системой, (например, сокращение затрат на поддержку старой и неэффективной системы) и пользу от новой нужной функциональности.

Также поищите существующую функциональность, которую не нужно сохранять в системе, которая придет на смену. Не дублируйте недостатки существующей системы и не упускайте возможности обновить систему для обслуживания новых бизнес-потребностей и бизнес-процессов. Например, бизнес-аналитик может спросить пользователей: «Используете ли вы *<название команды меню>*?» Если ответ всегда один: «Никогда не использовал», тогда, может, эта команда в новой системе не нужна. Изучите данные об использовании, показывающие, к которым окнам, функциям или сущностям данных в текущей системе обращаются очень редко. Даже существующей функциональности должны соответствовать текущие или будущие бизнес-цели, чтобы была причина повторно реализовывать ее в новой системе.

> **Примечание** Не позволяйте заинтересованным лицам использовать аргументацию типа: «Это есть сейчас, поэтому оно должно быть в новой системе» в качестве метода по умолчанию для оправдания требований.

Осторожно с пробелами

Анализ расхождений (gap analysis) — это сравнение функциональности между существующей и желаемой новой системой. Анализ расхождений можно представлять в разных формах, в том числе в виде вариантов использования, пользовательских историй и функций. При доработке существующей системы выполняйте анализ расхождений, чтобы убедиться, что понимаете, почему текущая система не соответствует бизнес-целям.

Анализ расхождений для проекта замены предполагает понимание существующей функциональности и выявление новой требуемой функциональности (рис. 21-1). Определите пользовательские требованиям к существующей системе, новую реализацию которых заинтересованные лица хотят видеть в новой системе. Также определите пользовательские требования, которым существующая система не удовлетворяет. Учтите запросы на изменения, которые никогда не реализовывались в существующей системе. Расставьте приоритеты существующих и новых пользовательских требований в совокупности. Определите приоритеты устранения расхождений с использованием бизнес-целей, как описано в предыдущем разделе, или других приемов определения приоритетов, представленных в главе 16.

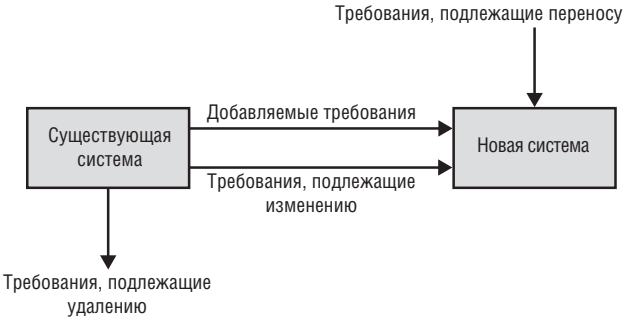


Рис. 21-1. При замене существующей системы часть требований реализуются без изменений, часть требует изменений, а еще есть требования, которые надо удалить, и новые требования, которые надо добавить

Сохранение уровня производительности

Существующие системы формируют у пользователей определенные ожидания в отношении производительности и пропускной способности. У заинтересованных лиц всегда есть ключевые индикаторы производительности (key performance indicators, KPI) существующих процессов, которые они хотят сохранить в новой системе. Модель ключевых индикаторов производительности (key performance indicator model, KPIM) может помочь выявить и определить эти метрики для соответствующих бизнес-процессов (Beatty and Chen, 2012). KPIM помогает заинтересованным лицам увидеть, что несмотря на то, что новая система будет другой, бизнес-результаты должны быть как минимум не хуже.

Если явно не заботиться об уровне производительности, по мере развития системы ее уровень упадет. Добавление новой функциональности в существующую систему может замедлить ее работу. У одного средства синхронизации было требование: обновлять основной набор данных путем добавления транзакций за день. Оно должно запускаться каждые 24 часа. В первом выпуске этого инструмента синхронизация началась в полночь и заняла около часа. После некоторых улучшений по включению дополнительных атрибутов, слияния и проверки синхронности, на синхронизацию стало уходить 20 часов. Это стало проблемой, потому что пользователи ожидали завершения ночной синхронизации до начала рабочего дня в 8:00. Максимальное время выполнения синхронизации никогда явно не указывалось, но заинтересованные лица предполагали, что оно должно выполняться ночью и занимать не более восьми часов.

При замене системы нужно расставить приоритеты коэффициентов KPI, которые важнее всего поддерживать. Выявите бизнес-процессы, связанные с самыми важными KPI, и требования, обеспечивающие реализацию этих бизнес-процессов, — такие требования нужно реализовывать в первую очередь. Например, в случае замены системы приема заявок на кредиты, в которой операционисты могут вводить 10 кредитов в день, в новой системе важно обеспечить как минимум такую же пропускную способность. В новой системе функциональность ввода кредитов должна реализовываться в первую очередь, чтобы не нарушать работу операционистов.

Когда старых версий требований нет

В большинстве старых систем требования вообще не задокументированы — об их точности вообще не стоит и заикаться. При отсутствии надежной документации по требованиям, разработчикам приходится заниматься обратной инженерией, т. е. разбираться в функциях системы исходя из ее интерфейсов, кода и содержимого баз данных. Я называю такую работу «археологией программного обеспечения». Чтобы она принесла максимум пользы, «археологическая экспедиция» должна документировать результаты своих изысканий в форме описания требований или дизайна. Накопив точную информацию об определенных частях текущей системы, команда сможет безопасно доработать систему, заменить систему, не потеряв критически важной функциональности и эффективно выполнить доработку. Прекращается непроизводительное растрачивание информации, и в будущем специалисты по обслуживанию смогут лучше разобраться в последних внесенных изменениях.

Если обновление требований обременительно, то им часто будут пренебрегать, поскольку занятые люди будут стараться сразу перейти к следующему запросу на обновление. А устаревшие требования вряд ли будут полезными при доработке в будущем. В отрасли разработки ПО бытует страх того, что на написание документации всегда уходит слишком много времени, поэтому вырабатывается условный рефлекс — пренебрегать всяким обновле-

нием документации к требованиям. Однако какова цена того, что требования *не обновляются* и специалисту по обслуживанию (а, может, им станете вы сами!) придется восстанавливать информацию? Ответив на этот вопрос, вы сможете принять разумное решение о том, следует ли обновлять документацию требований при разработке с нуля или изменении ПО.

При доработке или обслуживании системы команда сможет расширять эти раздробленные представления данных, таким образом, непрестанно улучшая документацию к системе. Дополнительные затраты на документирование этой вновь найденной информацией невелики по сравнению с ценой, которую кому-то придется заплатить в будущем, чтоб «открыть» ее заново. При реализации доработок практически всегда требуется разработка требований, поэтому нужно не забыть добавить эти новые требования в хранилище существующих требований, если такое хранилище существует. При замене старой системы есть возможность задокументировать требования к новой системе и поддерживать их в актуальном состоянии на протяжении всего проекта. Постарайтесь оставить требования в более привлекательном виде, чем вы их изначально нашли.

Какие требования нужно определять

Разработка полной спецификации требований к ПО для всей производственной системы не всегда целесообразна. Между двумя крайностями — работой безо всякой документацией и воссозданием идеальной спецификации — существует множество промежуточных стадий. Если вы знаете, зачем вам нужны задокументированные требования, то сможете решить, будут ли оправданы затраты на воссоздание всей спецификации или ее части.

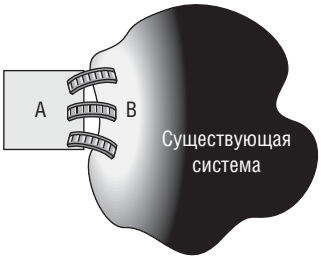


Рис. 21-2. Доработка A плохо задокументированной существующей системы предоставляет некоторую информацию об области B

Возможно, ваша текущая система представляет собой бесформенный сгусток истории и загадок, как на рис. 21-2. Представьте, что вас попросили реализовать некую новую функциональность в области A на этом рисунке. Начните с документирования новых требований в структурированной, пусть и неполной, спецификации требований к ПО или воспользуйтесь утилитой для управления требованиями. Когда вы добавите новую функциональность, вам придется сообразить, как новые экраны и функции будут вписываться

и взаимодействовать с существующей системой. Эти взаимодействия показаны на рис. 16-1 в виде мостов между областью A и текущей системой. Анализируя ситуацию, вы получите представление о белой области текущей системы — области B. Так вы выявите не только требования для области A, но и новую информацию, которую необходимо зафиксировать.

Документировать всю существующую систему приходится нечасто. Сосредоточьте свои усилия по детализации требований на изменениях, необходимых для достижения бизнес-целей. В случае замены системы начните с документирования областей, которые в процессе приоритизации определены как важные для достижения бизнес-целей или у которых самых высокий риск реализации. Все новые требования, выявленные в процессе анализа расхождений, нужно определить с таким же уровнем детализации и с применением тех же приемов, что используются в новой системе.

Уровень детализации

Одна из самых сложных задач — определение правильного уровня детализации документирования требований, сформулированных в процессе анализа существующих систем. При доработке определения только новой функциональности может быть достаточно. Однако обычно много пользы приносит также документирование всей функциональности, связанной с доработкой, чтобы изменения прошли без проблем (область B на рис. 21-2). Неплохо создать бизнес-процессы, пользовательские требования и/или функциональные требования для этих областей. Например, представьте, что при добавлении функции скидок в существующую корзину покупок у вас нет задокументированных требований к корзине. Вам может захотеться написать простую пользовательскую историю: «Как клиенту мне нужно ввести код скидки, чтобы получить самую низкую цену на выбранный товар». Однако в этой пользовательской истории отсутствует контекст, поэтому нужно постараться зафиксировать другие пользовательские истории операций с корзиной для покупок. Эта информация может оказаться ценной в следующий раз, когда вам потребуется изменить функцию корзины.

Как-то я работал с командой, которая только что начала заниматься требованиями для версии 2 крупного продукта со встроенным ПО. Требования для версии 1, которая в тот момент реализовывалась, были проработаны не очень хорошо. Ведущий бизнес-аналитик думал, стоит ли возвращаться и исправлять требования для версии 1. Руководство полагало, что это семейство продуктов будет серьезным источником дохода на протяжении как минимум следующих 10 лет, а также планировало повторно воспользоваться некоторыми основными требованиями в нескольких побочных продуктах. В этом случае имело смысл улучшить документацию требований для версии 1, поскольку она лежала в основе всей последующей работы над продуктами этого семейства. Если бы команда в тот момент работала над версией 5.3 предыдущей системы, которую планировалось изъять из производства в течение года, тогда воссоздавать спецификацию требований к ПО не стоило бы.

Данные отслеживания связей

Данные отслеживания требований помогут программисту, ведущему доработку, определить, какие компоненты следует модифицировать после изменения того или иного требования. В идеальном мире при замене системы есть полный набор функциональных требований к существующей системе, позволяющий отследить связи между старой и новой системами, чтобы не пропустить никаких требований. Однако в плохо задокументированной старой системе информации для отслеживания нет, а создание четкой системы отслеживания как в существующей, так и новой системах требует много времени.

Как и при любой другой разработке, хорошо позаботиться о создании матрицы отслеживания связей, которая увяжет новые или измененные требования с соответствующими элементами дизайна, кодом и тестами. Собирать связи по мере разработки не сложно, тогда как воссоздать их в завершенной системе — практически нереально. При замене систем отслеживайте связи требований на высоком уровне: создайте список функций и пользовательских историй для существующей системы и определите их приоритеты, чтобы можно было решить, какие из них реализовывать в новой системе. Подробнее об отслеживании требований см. главу 29.

Как собирать требования к существующей системе

Даже при отсутствии какой-либо документации в проектах доработки или замены есть система, которую можно исследовать для выявления требований. При работе над проектами замены стоит нарисовать карту новых экранов, которые вы собираетесь добавить, показав перемещения от существующих элементов экрана и к ним. Можно также записать варианты использования или пользовательские истории, охватывающие новую и старую функциональность.

В проектах замены системы нужно понимать всю требуемую функциональность — точно так же, как в проектах разработки нового продукта. Изучите интерфейс существующей системы, чтобы определить функциональность для реализации в новой системе. Проанализируйте интерфейсы существующей системы, чтобы понять, каким данными обмениваются системы сейчас. Разберитесь, как пользователи работают с текущей системой. Если никто не понимает функциональность и бизнес-правила, стоящие за интерфейсом, кому-то придется проанализировать код или базу данных, чтобы понять, что происходит. Для выявления требований проанализируйте всю имеющуюся документацию — документы дизайна, страницы справочной системы, руководства пользователя, обучающие материалы.

Может вообще оказаться ненужным определять функциональные требования для существующей системы, будет достаточно создать модели, чтобы заполнить информационный пробел. Диаграммы swimlane могут описывать, как пользователи выполняют свою работу сейчас. Полезны также контекстные диаграммы, диаграммы потоков данных и диаграммы «сущность–связь». Можно создать пользовательские требования, определив их только

на высоком уровне, не описывая подробности. Еще один способ получения недостающий информации — добавлять в словарь данных новые понятия и изменять существующие определения при добавлении новых и модификации существующих элементов системы. Набор тестов может пригодиться в качестве первоначального источника информации при восстановлении требований к ПО, потому что тесты являются альтернативным представлением требований.

Иногда «нормально» бывает недостаточно

Оценка сторонней организацией текущих процессов бизнес-аналитики в компании показала, что команды хорошо справляются с написанием требований для новых проектов, но не обновляют требования по мере развития продуктов в последующих выпусках. Бизнес-аналитики создали требования для всех проектов доработки, но не внесли соответствующие обновления в базовую версию требований. Менеджер организации не видел пользы от обеспечения стопроцентной актуальности существующей документации, отражающей состояние внедренных систем. Он предполагал, что его требования все равно всегда отражали только 80-90% работающего ПО, поэтому нет особого смысла стараться доводить до идеала требования при доработке. Это означало, что в будущих проектах доработки командам придется мириться с некоторой неопределенностью и закрывать пробелы по мере необходимости, но затраты на это считались допустимыми.

Продвижение новой системы

При изменении или замене существующей системы вы скорее всего столкнетесь с сопротивлением. Люди не очень любят перемены. Добавление новой функции, которая облегчит работу пользователей, это благо. Пользователи привыкли, как система работает сейчас, а вы планируете изменить ее, что с точки зрения пользователя не очень хорошо. Ситуация значительно усложняется, если вы заменяете систему, потому что теперь вы меняете больше чем какую-то часть функциональности. Вы можете поменять поведение и вид всего приложения, его меню, рабочую среду и, возможно, всю работу пользователя. Если вы бизнес-аналитик, менеджер или куратор проекта, нужно учитывать сопротивление и планировать, как его преодолеть, чтобы пользователи приняли новые функции или систему.

Существующая, установившаяся система скорее всего стабильна и полностью интегрирована с окружающими системами, а пользователи ее хорошо понимают. В первом выпуске новой системы с той же функциональностью всего этого может не быть. Пользователи могут бояться, что новая система нарушит их обычную работу, а еще им придется тратить усилия на ее освоение. Может быть и хуже — она не сможет обеспечить поддержку их текущих операций. Пользователи могут даже бояться потерять место, если система автоматизирует задачи, которые они сейчас выполняют вручную. Очень ча-

сто от пользователей приходится слышать, что они примут новую систему, только если она будет делать все то же, что делала старая, — даже если они сами сейчас не используют всю эту функциональность.

Для снижения риска сопротивления первым делом надо понять бизнес-цели и пользовательские требования. Если эти вещи не выполняются, вы быстро потеряете доверие пользователей. Во время выявления требований надо сосредоточиться на преимуществах новой системы или каждой функции, которую вы предоставите пользователям. Помогите им понять ценность предлагаемых изменений для организации в целом. Не забывайте, что даже при внесении улучшений «новое» не обязательно означает «облегчающее работу пользователя». Плохо спроектированный пользовательский интерфейс может даже усложнить работу с системой, потому что становится сложнее найти старые функции, потерявшись в джунглях новых параметров, или доступ к ним затрудняется.

В нашей компании недавно обновили систему хранения документов до новой версии, предоставив дополнительные возможности и более стабильную рабочую среду. Во время бета-тестирования я обнаружил, что усложнились простые, стандартные задачи, такие как блокировка для редактирования (check-out) и загрузка файла. В предыдущей версии блокировка выполнялась двумя щелчками, а теперь их число увеличилось до трех или четырех. Если бы наши исполнительные заинтересованные лица посчитали такие изменения пользовательского интерфейса рискованным с точки зрения принятия, они бы потратили силы на разработку нестандартной функциональности, повторяющей работу старой системы. Демонстрация прототипов пользователям могла бы помочь им привыкнуть к новой системе или новым функциям, а также выявить проблемы с принятием новой системы на ранних стадиях проекта.

Есть одна оговорка касательно замены системы: в определенных группах она может оказать отрицательное влияние на ключевые индикаторы производительности, даже если от замены системы выиграет организация в целом. Как можно раньше сообщите пользователям о функциях, которые скорее всего будут удалены, или об атрибутах качества, которые могут пострадать, — это позволит им подготовиться к переменам. Принятие системы основано как на эмоциях, так и на логике, поэтому управление ожиданиями критически важно для создания фундамента для успешного развертывания.

При переходе от существующей системы важны также переходные требования. Они описывают возможности всего решения, а не только программного приложения, и должны обеспечивать переход от существующей к новой системе (IIBA, 2009). Они могут предусматривать преобразование данных, обучение пользователей, изменения в организационных и бизнес-процессах, а также параллельную работу старой и новой систем на протяжении какого-то времени. Подумайте том, что может потребоваться заинтересованным лицам для комфортного и эффективного перехода на новые «рельсы». Понимание переходных требований является частью оценки готовности и управления организационных изменений (IIBA, 2009).

Уместны ли итерации

Проекты доработки по определению являются инкрементальными. Проектные команды часто используют методы гибкой разработки (agile), определяя приоритеты доработок с применением резерва (backlog) проекта, как описано в главе 20. Однако проекты замены не всегда подходят для поэтапного, итеративного развертывания — нужна критическая масса функциональности нового приложения, прежде чем пользователи смогут применять его для выполнения своей работы. Им нет смысла использовать новую систему для выполнения небольших частей своей работы, а затем возвращаться в старую систему для выполнения других функций. Вместе с тем миграции типа «все и сразу» также сложны и нереалистичны. Сложно за раз заменить установившуюся систему, которая развивалась много лет и претерпела много выпусков.

Один из способов поэтапной реализации системы на замену — определить функциональность, которую можно изолировать, и разрабатывать только такие части. Однажды мы помогали команде у клиента заменять их текущую систему на новую. На управление складом приходилось 10% всей функциональности системы. По большей части сотрудники, управляющие запасами, работали отдельно от тех, что занимались другими бизнес-процессами. Исходная стратегия заключалась в переносе в новую систему только функциональности склада. Эта функциональность отлично подходила для изоляции в первом выпуске, потому что влияла лишь на часть пользователей, которые в дальнейшем работали в основном в новой системе. Но были и сложности: нужно было разработать новый интерфейс для обмена данными между новой системой склада и существующей системой.

У нас не было никакой документации по требованиям для существующей системы, но сохранение исходной системы и отключение в ней функциональности склада позволило четко задать границы для сбора требований. Для новой системы склада мы в основном писали варианты использования и функциональные требования, беря в качестве основы важные функции существующей системы. Мы создали диаграмму «сущность–связь» и словарь данных. Мы нарисовали контекстную диаграмму для всей системы, чтобы понимать, где находятся точки интеграции, о которых нам нужно знать при отсечении склада. Далее мы создали новую контекстную диаграмму, чтобы показать, как управление складом будет существовать в виде внешней системы с урезанной внешней системой.

Не во всех проектах улучшения или доработки бывает все так гладко. В большинстве из них приходится решать две самые большие проблемы: отсутствие документации по существующей системе и возможность борьбы за то, чтобы пользователи приняли новую систему или функции. В любом случае применение описанных в этой главе приемов помогает активно снижать подобные риски.

Глава 22

Проекты с серийным продуктом

Некоторые организации для удовлетворения своих потребностей в ПО приобретают и адаптируют *пакетные решения* (их еще называют тиражируемыми, серийными или готовыми продуктами), предпочитая не разрабатывать новых систем с нуля. Также в последнее время приобрели популярность SaaS (Software As A Service), или облачные решения. Даже при покупке готового пакета в качестве части или всего решения для нового проекта или при реализации решения в облаке, вам понадобятся требования. Требования позволяют оценивать решения-кандидаты и выбирать самый подходящий пакет, а также адаптировать пакет в соответствии с вашими потребностями.

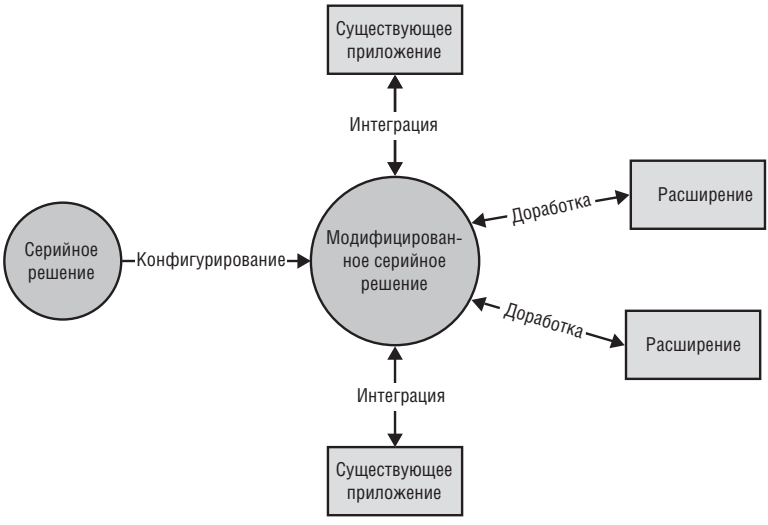


Рис. 22-1. Готовые пакеты можно конфигурировать, интегрировать в существующую прикладную среду и расширять, добавляя новую функциональность

Как показано на рис. 22-1, готовые продукты, как правило, нужно конфигурировать, настраивать, интегрировать и расширять для работы в целевой среде. Некоторые серийные продукты практически готовы и для их развертывания и использования не требуется прилагать никаких дополнительных усилий, но есть и такие, которые требуют определенной настройки. Это мо-

жет быть конфигурирование продукта по умолчанию, интеграция с другими системами и/или разработка расширений, обеспечивающих дополнительную функциональность, которой нет в серийном продукте. Для всего этого необходимы требования.

В этой главе описываются требования к выбору и реализации пакетных решений. Мы не различаем серийные и SaaS-проекты по причине похожести работы с требованиями. Решение о внедрении пакетного решения, а не разработке новой системы, основывается на оценке затрат на эти два варианта и выходит за рамки этой книги. Если вы создаете пакетное решение на продажу, вам нужно обратиться к другим главам этой книги, потому что такие проекты предусматривают разработку ПО.

В этой главе описывается несколько способов определения требований при планировании приобретения коммерческого продукта, отвечающего вашим потребностям. Мы также предоставляем советы, как разрабатывать требования при реализации серийного решения в вашей рабочей среде.

Требования к выбору тиражируемых решений

При приобретении готовых пакетов организация получает меньшую гибкость при работе с требованиями, чем при разработке продукта «под заказ». Вам необходимо выяснить, какие из запрошенных возможностей не могут быть предметом переговоров, а какие вы может изменять в рамках ограничений, налагаемых пакетом. Единственная возможность выбрать правильный пакет решений — понять бизнес-операции, для решения которых вы приобретаете пакет. Выбор пакетных решений подразумевает определение своих требований к ПО, по крайней мере на высоком уровне. Уровень детализации и усилий, прилагаемых к определению требований для выбора серийного продукта, зависит от ожидаемой стоимости пакета, отведенного на оценку времени и числа решений-кандидатов. Просто сравните покупку программы для ведения личных финансов и приобретение многомиллионного финансового приложения для компании с 5000 сотрудниками. В первом случае достаточно перечислить самые важные варианты использования, а во втором потребуется написать полные варианты использования и разработать требования к данным и качеству для более углубленной оценки.

Одной команде нужно было выбрать серийное ПО для поддержки юридической компании. Команда определила 20 задач, которые нужно выполнять пользователям с помощью этого ПО, в результате были сформулированы 10 функций, что, в свою очередь, позволило ограничить круг кандидатов до четырех пакетов. Владельцы фирмы знали, что после выбора пакета им потребуется создать более подробные требования по конфигурированию ПО. Однако для выбора пакета подошла более легкая оценка. С другой стороны, команда из 50 человек работала над разработкой подробных требований к

ПО для обслуживания завода полупроводниковых приборов. Они оценивали только три решения-кандидата, но ввиду высокой стоимости приобретения и внедрения тиражируемого ПО компания была готова потратить значительные средства на процесс выбора. Только на выбор решения было затрачено полгода.

Разработка пользовательских требований

Любой выбранный вами пакет должен обеспечивать выполнение пользовательских задач, хотя каждый пакет делает это своим способом. При работе над большей частью требований по приобретению серийного продукта стоит обратить особое внимание на уровень требований пользователей. И здесь вам помогут варианты использования продукта и пользовательские истории. Можно также использовать модели процессов, или они уже могут иметься в организации. Нет особого смысла разрабатывать подробные функциональные требования или пользовательский интерфейс, если вы планируете приобрести готовый продукт — поставщик сделал это за вас.

Бывает также полезно составить список функций, которые требуются от пакетного решения. Определите требуемые функции продукта, используя свое понимание того, что пользователи должны получить от решения и какие бизнес-процессы оно должно обеспечивать. Представьте, что ваша пользовательская история выглядит так: «Как менеджеру по исследованиям мне нужно проверять и одобрять новые эксперименты до их выполнения, чтобы не тратить время и материалы на некачественные эксперименты» Эта пользовательская история позволяет обнаружить потребность в функции, реализующей процесс одобрения.

Вряд ли вы найдете готовые решения для каждого варианта использования, поэтому расставьте их и сообщения по приоритетам. Свяжите их с бизнес-требованиями, чтобы не тратить время на ненужные критерии оценки. Определите, какие функции должны быть доступны в первый день, какие могут подождать до дальнейших расширений и без каких пользователи не смогут работать.

Работа с бизнес-правилами

Изучая требования, необходимо определить бизнес-правила, которым готовый продукт должен соответствовать. Сможете ли вы сконфигурировать продукт в соответствии с корпоративной политикой, промышленными стандартами или постановлениями регулирующих органов? Насколько легко вносить коррективы при изменении этих правил и положений? Сосредоточьтесь на самых важных бизнес-правилах, потому что на оценку реализации всех правил может потребоваться слишком много времени.

Некоторые пакеты учитывают глобальные бизнес-правила, например расчет подоходного налога с дохода или печать деклараций о доходах. Уверены ли вы, что они реализованы корректно? Обеспечит ли вас поставщик обнов-

ленной версией пакета в случае изменения этих правил и расчетов? Будут ли обновления бесплатными? Предоставит ли поставщик список бизнес-правил, реализованных в пакете? Если продукт поддерживает какие-либо бизнес-правила, которые для вас неприемлемы, можете ли вы отключить, изменить или обойти их? Принимает ли поставщик запросы на улучшения? Если да, то как определяются их приоритеты?

Определение потребностей в данных

Вы также можете определить структуры данных, необходимые для удовлетворения пользовательских требований и бизнес-правил, особенно если новое решение должно интегрироваться в экосистему существующих приложений. Попробуйте отыскать основные несоответствия между вашей моделью данных и моделями данных поставщика. Не обращайте внимания на сущности и атрибуты данных, которые в серийном продукте названы иначе. Вместо этого постарайтесь найти сущности и их атрибуты, которых нет в серийном решении или определение которых сильно отличается от того, что нужно вам, а затем определите, можно ли с этими сущностями работать иначе так, чтобы решение работало.

Определите отчеты, которые должно создавать серийное решение. Генерируются ли необходимее отчеты в правильных форматах? Насколько продукт позволяет модифицировать стандартные отчеты? Можно ли проектировать собственные отчеты для интеграции с теми, что предоставляет поставщик?

Определение требований по качеству

Атрибуты качества, о которых рассказывалось в главе 14, — еще один жизненно важный аспект требований пользователей, который имеет большое значение при выборе пакетного решения. Следует рассмотреть, по крайней мере, следующие атрибуты.

- **Производительность** Каково максимальное время отклика, приемлемое для конкретных операций? Может ли пакет обработать ожидаемое количество одновременно подключенных пользователей и транзакций?
- **Легкость и простота использования** Соответствует ли пакет всем принятым соглашениям пользовательского интерфейса? Похож ли интерфейс на интерфейсы других приложений, с которыми знакомы пользователи? Насколько легко ваши клиенты смогут обучиться работать с новым пакетом? Предоставляет ли поставщик обучение и входит ли оно в стоимость пакета?
- **Гибкость** Насколько легко разработчики смогут изменить или расширить пакет, чтобы удовлетворить те или иные потребности? Входят ли в пакет соответствующие «зацепки» (точки подключения и расширения) и API- интерфейсы, чтобы добавлять расширения? Останутся ли все такие расширения на месте при установке новой версии пакета?

- **Способность к взаимодействию** Насколько легко вы сможете интегрировать пакет с другими корпоративными приложениями? Не заставит ли он вас обновить какие-либо инструменты или компоненты инфраструктуры, потому что не обеспечивает обратной совместимости?
- **Целостность** Предохраняет ли пакет данные от потери, повреждения или неправомочного доступа?
- **Безопасность** Обеспечивает ли пакет управление доступом пользователей к системе или отдельным ее функциям? Можно ли определить необходимые уровни привилегий пользователей? В отношении SaaS-решений надо очень внимательно оценить соглашения об уровне обслуживания с оглядкой на ваши требования.

Оценка решений

Многие пакеты предоставляют готовые решения для удовлетворения определенной части нужд, связанных с обработкой информации. Выполните начальный анализ рынка и отберите пакеты-кандидаты, заслуживающие дальнейшего рассмотрения. После этого можно использовать требования, определенные вами как критерий оценки, в процессе информированного выбора серийного ПО.

Один из подходов к оценке предусматривает следующую последовательность действий (Lawlis и другие, 2001):

1. Оцените свои требования по шкале от 1 до 10, чтобы определить важность каждого.
2. Определите, насколько каждый пакет-кандидат соответствует каждому требованию. Используйте оценку «1» для обозначения полного удовлетворения, «0,5» — для частичного удовлетворения и «0» — если вас все не устраивает. Вы можете найти информацию, чтобы выполнить оценку, в документации по продукту, в ответе поставщика на запрос на предложение (приглашение принять участие в торгах по проекту) или с помощью прямой проверки продукта. Помните, что запрос на предложение является приглашением принять участие в торгах по проекту и может не соответствовать тому, как вы планируете использовать продукт. Прямая проверка необходима для высокоприоритетных требований.
3. Подсчитайте общую сумму очков для каждого пакета-кандидата, сложив очки, которые вы определили для каждого пункта, и вы поймете, какие продукты подходят вам лучше всего.
4. Оцените стоимость продукта, опыт и надежность поставщика, поддержку продукта поставщиком, внешние интерфейсы, которые позволят осуществлять расширение и интеграцию продукта, а также совместимость с технологическими требованиями или ограничениями для вашей среды. Стоимость является критерием отбора, но сначала кандидатов нужно оценивать без учета цены.

Можно посмотреть, какие требования *не удовлетворяются* какими-то пакетами-кандидатами, что потребует разработки расширений. Это может серьезно повысить стоимость внедрения серийного продукта, поэтому должно учитываться в процессе оценки.

Недавно в моей организации решили выбрать средство управления требованиями, которое помимо остальных возможностей позволяло бы пользователям работать в отсутствие интернет-подключения и выполняло бы синхронизацию с основной версией требований при восстановлении связи (Beatty and Ferrari, 2011). Мы подозревали, что на рынке нет хороших решений, предоставляющих такую функцию. Мы включили эту функцию в оценку, чтобы не забыть о ней при отборе решения. Если мы не найдем решения с такой функцией, мы будем знать, что ее придется реализовывать как расширение существующего пакета. Альтернативный вариант — изменить процесс редактирования требований.

Идентификатор	Вариант использования	Функция	Приоритет	Средство 1	Комментарии к средству 1
1	Бизнес-аналитик добавляет новые требования по одному	Добавление нового требования	10	1	
2	Бизнес-аналитик добавляет новые требования по одному	Автоматическое создание уникального идентификатора для каждого требования	10	1	
3	Бизнес-аналитик добавляет новые требования по одному	Документирование требований в виде текста с форматированием	3	0	Поддерживает только текст без форматирования
5	Бизнес-аналитик моделирует требование	Описание требования с помощью рисунка непосредственно в базе данных	6	0,5	Требуется обходное решение: в базе данных размещается ссылка на внешний ресурс
6	Бизнес-аналитик моделирует требование	Описание требования в виде встроенного документа в базе данных	8	1	
7	Бизнес-аналитик связывает существующую документацию с требованием	Создание ссылки, связывающей требования с документами в SharePoint	4	1	
9	Бизнес-аналитик добавляет целый набор требований за раз	Пакетный импорт структурированных данных как новых требований из Excel	5	1	Пакетный импорт поддерживается и обеспечивает поддержку импорта нестандартных файлов Excel

Рис. 22-2. Пример матрицы оценки пакетных решений для управления требованиями

Другой вариант определения, позволит ли пакет пользователям выполнять их задачи, и если да, то насколько хорошо, — создание тестов на основе высокоприоритетных вариантов использования. Включите в них тесты, в которых анализируется, как система обрабатывает возможные важные условия исключений. Пройдите эти тесты, чтобы узнать, как пакеты-кандидаты справляются с ними. Схожий прием — прогнать готовый продукт через набор сценариев, представляющий ожидаемые модели использования, — их еще называют оперативным профилем (Musa, 1999).

> **Внимание!** Если у вас нет хотя бы одного человека, который участвует во всех оценках, нет никаких гарантий, что вы применяете сопоставимые интерпретации функций и очков.

В результате процесса оценки обычно получают матрицу оценки с требованиями выбора в строках и очков различных решений в столбцах. На рис. 22-2 показана часть примера матрицы оценки средства управления требованиями.

Многоэтапная оценка

Когда я писал требования для выбора средства управления требованиями для наших команд консультантов, я работал с командами над определением классов пользователей и вариантов использования. Основными пользователями были бизнес-аналитики, но были также варианты использования для менеджеров, разработчиков и клиентов. Я определял варианты использования по имени и использовал свое знание этих вариантов при формулировке требуемых функций. Я создал матрицу связей, чтобы свести к минимуму возможность упустить какие-то варианты использования или функции.

Мы начали с 200 функций и 60 поставщиков, что совсем не укладывалось в наши временные рамки. Мы провели первый этап оценки, чтобы отсеять большинство кандидатов. На первом этапе мы анализировали только 30 функций, которые считали самыми важными и отличающими средства друг от друга. В результате поле поиска сузилось до 16 кандидатов. После этого мы оценили их по 200 функциям. В результате второго этапа мы получили список из пяти очень близких по функциональности инструментов, которые четко соответствовали нашим потребностям.

Помимо объективного анализа, очень хорошо оценить кандидатов, задействовав их в реальном проекте, а не просто в учебном проекте, предоставляемом в составе продукта. Мы организовали третий этап оценки, на котором попробовали каждый из пяти инструментов в деле на реальных проектах, чтобы увидеть, насколько их реальные показатели соответствуют нашей оценке. Третий этап оценки позволил выбрать предпочтительный инструмент из кандидатов с наивысшими оценками.

Требования к внедрению серийных решений

Принятие решения о внедрении определенного серийного решения — только часть задачи: нужно еще поработать над требованиями. На рис. 22-3 показано, что объем усилий, который нужно приложить к тому, чтобы пакетное решение принесло пользу, меняется в зависимости от типа внедрения: от использования решения «как есть» без изменений до выполнения значительной работы по спецификации требовании и разработке программных расширений. В табл. 22-1 описаны четыре типа внедрения пакетного решения, которые не являются взаимоисключающими. Любой из перечисленных типов внедрения может требовать внесения изменений в рабочую среду, например обновления операционных систем или других программных компонентов, взаимодействующих с пакетом.

Рис. 22-3. Объем работ для разных пакетных решений

Табл. 22-1. Подход к внедрению серийных продуктов

Тип внедрения	Описание
Продукт без изменений	Установка ПО без внесения каких-либо изменений в него
Конфигурирование	Настройка параметров ПО в соответствии с потребностями организации без разработки какого-либо кода
Интеграция	Подключение пакета к существующим системам в вашей экосистеме. Обычно требуется разработка небольшого объема специального кода
Расширение и доработка	Разработка дополнительной функциональности для расширения возможностей, отсутствующих в стандартном пакете

Одно из преимуществ приобретения готового решения в том, что оно может содержать возможности, о которых вы изначально не думали. Пакет обычно выбирают, основываясь на том, что знают. Но в процессе внедрения могут обнаружиться ценные возможности, о которых и подумать не могли. Это может изменить объем работы по установке пакета так, чтобы можно было воспользоваться дополнительными возможностями.

Требования к конфигурированию

Иногда пакет можно использовать в том виде, в котором он пришел от поставщика. Но чаще приходится «подкручивать» различные параметры конфигурации, чтобы пакет лучше решал поставленные задачи. Требования к конфигурированию жизненно важны для большинства проектов по внедрения серийного решения. Один из подходов заключается в определении требований к конфигурации одного потока процессов, варианта использования или пользовательской истории за раз. Изучите руководства пользователя для приобретенной системы, чтобы узнать, как выполнять ту или иную задачу, стараясь выяснить, какие параметры конфигурации подойдут вашей среде. При настройке системы учитывайте не весь набор бизнес-правил, а только те, которые изучались при выборе системы. Может быть полезным создать таблицу решений и деревья решений, моделирующие эти требования. Многие серийные решения поставляются со встроенными механизмами для определения ролей и разрешений. С помощью матрицы ролей и разрешений, такой как показано на рис. 9-2 в главе 9, определите, какие надо создать роли и какие им назначить разрешения.

Требования по интеграции

Пакетное решение нужно интегрировать в существующую прикладную среду, если только оно не используется в автономном режиме. Интеграция предусматривает понимание внешних интерфейсов, предоставляемых пакетом каждому из приложений, с которыми он будет взаимодействовать. Точно определяйте требования к обмену данными и сервисами между пакетом и другими компонентами среды. Скорее всего потребуется написать дополнительный код, чтобы обеспечить взаимодействие всех частей. Это могут быть:

- адаптеры, изменяющие интерфейсы и добавляющие отсутствующую функциональность;
- брандмауэры, изолирующие серийное ПО от других частей предприятия;
- оболочки, перехватывающие входные и выходные данные пакета и модифицирующие данные, как того требуется на другой стороне интерфейса (NASA, 2009).

Требования по расширению

Общая цель всех проектов по развертыванию серийных решений — свести к минимуму модификацию решения. На другом конце спектра — создание приложения своими силами. Но в большинстве таких проектов существуют пробелы между потребностями организации и возможностям пакета. По каждому такому пробелу нужно принять решение: игнорировать его (удалить требование и использовать инструмент «как есть»), изменить что-то за пределами решения (изменить бизнес-процесс) или создать «мостик», закрывающий пробел (расширить решение). При расширении пакетного решения нужно полностью определить требования к новым возможностям — так же, как при разработке новых функций в новом продукте. При развертывании серийного решения взамен старой системы воспользуйтесь приемами по замене системы, описанными в главе 21. При анализе требований к любому добавляемому компоненту оцените, не окажет ли он отрицательное влияние на существующие элементы или процессы в пакете.

Требования к данным

Начните с требований к данным, которые использовались при выборе. Сопоставьте сущности и атрибуты существующего словаря данных с сущностями и атрибутами серийного продукта. Скорее всего будут области, в которых решение не поддерживает существующих сущностей и атрибутов. Как и с функциональными пробелами, нужно решить, как поступать с пробелами в данных, — обычно добавляют дополнительные атрибуты или приспосабливают существующие структуры данных серийного решения. В противном случае при преобразовании и переносе данных из существующих систем в пакетное решение вы скорее всего потеряете данные, для которых соответствие не установлено. Используйте таблицы отчетов для определения тре-

бований к развертыванию существующих или новых отчетов, как описано в главе 13. Многие серийные пакеты предоставляют шаблоны стандартных отчетов, с которых можно начать.

Изменение бизнес-процессов

Серийные решения обычно выбирают, ожидая, что их реализация и поддержка обойдутся дешевле, чем разработка своего ПО. Организации должны быть готовыми адаптировать свои бизнес-процессы к возможностям и ограничениям потоков процессов в решении. В этом отличие от большинства проектов разработки, где ПО разрабатывается специально для поддержки существующих и планируемых процессов. Стоит заметить, что серийное решение, конфигурация которого обеспечит полную поддержку существующих процессов, очень сложно и дорого. Чем больше кнопок и регуляторов нужно настроить, тем сложнее конфигурация. Нужно выбрать что-то среднее между реализацией всей нужной функциональности или только тем, что предлагает серийное решение «в штатной комплектации» (Chung, Hooper и Huynh, 2001).

Начните с требований к данным, которые использовались при выборе. Разработайте варианты использования и диаграммы swimlane, чтобы понять, как изменятся задачи, когда пользователи будут выполнять их в серийном решении. Пользователи могут сопротивляться новому пакетному решению, потому что оно выглядит и ведет себя иначе, чем существующие системы, — постарайтесь привлечь их к процессу на ранних стадиях. Пользователи более благосклонно принимают новое решение, если сами участвуют в корректировке своих бизнес-процессов.

В моей команде внедряли пакетное решение для страховой компании, обеспечивающее выполнение новых нормативных требований. Мы начали с моделирования существующих бизнес-процессов. После этого мы изучили пользовательскую документацию пакета, чтобы получить общее представление о том, как использовать продукт. На основе модели существующей системы мы создали будущие процессы в соответствии с тем, как пользователи будут выполнять свои задачи в серийном решении. Мы также создали словарь данных существующей системы и добавили столбец для соответствующего поля в серийном решении. Пользователи помогали в этой работе, поэтому они не были удивлены новой системой при ее развертывании.

Обычные сложности с пакетными решениями

Далее перечислены стандартные сложности и проблемы, возникающие при выборе и внедрении пакетного решения:

- **Слишком много решений-кандидатов** На вскидку может оказаться очень много решений на рынке, отвечающим вашим требованиям. Соз-

дайте сокращенный перечень критериев, чтобы ограничить список кандидатов небольшим количеством.

- **Слишком много критериев оценки** Бывает тяжело ограничиться только самыми важными критериями, не вдаваясь в глубокий анализ спецификаций требований. Использование бизнес-целей позволяет отобрать самые важные требования. Сократив перечень пакетов-кандидатов до небольшого их числа, можно перейти к оценке на основе длинного списка критериев.
- **Некорректное представление возможностей пакета поставщиком** В типичном процессе приобретения тиражируемого ПО агенты по продажам на стороне поставщика пытаются убедить в полезности решения лиц организации, ответственным за принятие решений, а затем подключается команда внедренцев, предоставляющая детальные сведения о продукте. Эти сведения могут немного не соответствовать тому, что клиент понял о возможностях продукта на основе общения с агентами по продажам. Очень удачная идея попросить технического специалиста поставщика поучаствовать на стадии продажи. Определите, сможете ли вы наладить хорошие деловые отношения с поставщиком, которые способствовали успеху обеих сторон. Поставщик — ваш партнер по бизнесу, поэтому нужно выстраивать конструктивные отношения.
- **Неверные ожидания от решения** Подчас при демонстрации поставщиком решение выглядит прекрасно, но после установки работает не так, как ожидалось. Чтобы избежать недоразумений, попросите поставщика продемонстрировать конкретные варианты использования, чтобы вы смогли увидеть, насколько решение отвечает вашим ожиданиям.
- **Пользователи отвергают решение** Одного факта приобретения ПО организацией не достаточно, чтобы пользователи приняли новое решение. Как и в проектах разработки ПО, привлекайте пользователей к процессу выбора или на ранних стадиях внедрения, чтобы гарантировать, что их потребности поняты правильно и удовлетворены по максимуму, насколько это возможно. Управление ожиданиями — важная часть внедрения пакетного решения.

Приобретение, конфигурирование и расширение серийного пакета ПО часто бывает разумной бизнес-альтернативой разработке собственного решения. Пакеты предоставляют большую гибкость, но одновременно в них есть пределы и ограничения. Не хочется платить за большое количество функций, которые не нужны организации. Или создавать хрупкую конструкцию из расширений и модулей интеграции, которая может рухнуть в следующем выпуске пакета поставщиком. Тщательный выбор пакета и продуманный процесс внедрения поможет найти оптимальный баланс между возможностями, удобством использования и обслуживания, а также расширяемостью пакетного программного решения.

Глава 23

Проекты, выполняемые сторонними организациями

Вместо того чтобы создавать системы самостоятельно, многие организации отдают разработку на сторону сторонним компаниями-подрядчикам, или, как еще говорят, прибегают к аутсорсингу (outsourcing). Организация может поручить разработку другой компании из-за отсутствия разработчиков, для расширения ресурсов разработки, для экономии денег или для ускорения разработки. Подрядчик может находиться рядом, на другом конце мира или где-то посередине между этими крайними случаями. Разработку, выполняемую командами в других странах, обычно называют оффшорным программированием.

Все аутсорсорснговые проекты предусматривают наличие распределенных команд, размещенных в двух или большем количестве мест. В таких проектах роль бизнес-аналитика особенно важна Часто работа бизнес-аналитика усложняется. Если все члены команды находятся в одном месте, разработчики могут просто подойти к бизнес-аналитику и задать вопрос или показать вновь созданную функциональность. Такое тесное взаимодействие невозможно при ауторсинговой разработке, хотя современные средства связи сильно помогают. В сравнении с «домашней» разработкой в аутсорсинговых, и особенно оффшорных, проектах возникают следующие сложности с требованиями:

- сложнее получить отклик о требовании от разработчика и передать отзывы пользователей предоставленного ПО разработчикам;
- требуется формальное контрактное определение требований, что может создавать сложности в случае обнаружения различий в интерпретации на поздних стадиях проекта;
- разрыв между тем, что хотели пользователи, и продуктом, который они получают в конечном итоге на основе начальных требований, потому что меньше возможностей скорректировать работы по ходу проекта;
- разрешение проблем с требованиями может занимать больше времени из-за размещения в разных часовых поясах;
- передача сведения о требований усложняется из-за языковых и культурных различий;

- лаконичных письменных требований, которые обычно достаточны в обычных проектах, в аутсорсинговых проектах недостаточно, а бизнес-аналитики не всегда доступны, чтобы ответить на вопросы разработчиков, разъяснить непонятные места и заполнить пробелы;
- удаленным разработчикам не хватает организационных и бизнес-знаний, которые локальные разработчики усваивают с опытом.

Хотя и считается, что оффшорная разработка позволяет снизить затраты за счет более использования более дешевой рабочей силы, на самом деле во многих оффшорных проектах затраты *больше*. Основными причинами являются дополнительные усилия на создание более точных требований, повышенная вероятность дополнительной разработки для заполнения пробелов из-за несформулированных, неявных и подразумеваемых требований, дополнительные расходы на согласование контрактных обязательств, начальные затраты на выработку эффективных норм поведения между командами, а также издержки, связанные с дополнительными коммуникациями и контролем в процессе выполнения проекта.

Чаще всего сторонним организациям поручают разработку ПО, но нередко на сторону передают и тестирование. С аутсорсингом тестирования возникают те же проблемы, что и с аутсорсингом разработки. Успех и того, и другого сильно зависит от качества и четкости требований.

В этой главе предлагаются самые важные приемы обеспечения успешной разработки требований и управления аутсорсинговыми проектами. В этой главе не обсуждается процесс принятия решения, нужно ли поручать разработку сторонней организации, или процесс выбора подрядчика.

Необходимый уровень детализации требований

Чтобы поручить разработку продукта другой компании, необходимо иметь требования высокого качества, поскольку, вероятнее всего, прямые контакты с командой разработчиков будут минимальными. Как показано на рис. 23-1, вы отправляете подрядчику запрос на предложение (RFP), спецификацию требований и критерии приемки продукта. Затем оба участника совместно анализируют ситуацию и достигают соглашения, скорее всего в процессе переговоров и корректировок, — только после этого подрядчик приступает к разработке. Далее подрядчик предоставляет готовое ПО и вспомогательную документацию.

При использовании аутсорсинга у вас не будет возможности для ежедневных уточнений, принятия решений и изменений, которые вы практикуете, когда разработчики и заказчики работают в непосредственной близости друг от друга. В частности при оффшорной разработке нужно надеяться, что подрядчик создаст точно то, что заказано. Вы получите не больше и не меньше заказанного, причем часто дополнительных вопросов не задают. Подрядчик

не станет реализовывать неявных и подразумеваемых требований, которые вы посчитали слишком очевидными, чтобы зафиксировать на бумаге. В результате плохо определенные и управляемые требования зачастую становятся причиной неудачи проекта, выполняемого сторонними организациями.

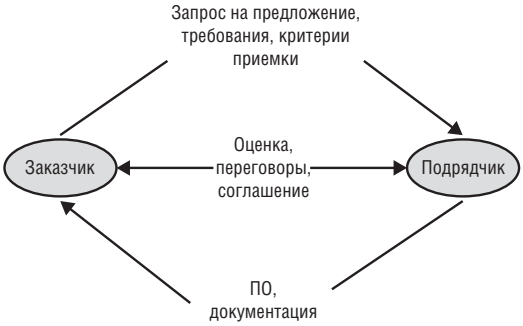


Рис. 23-1. Требования — краеугольный камень проекта аутсорсинга

Если вы распространяете запрос на предложение (RFP), поставщикам необходимо точно знать, что именно вы хотите, чтобы сообщить реалистичную информацию и оценку (Porter-Roth, 2002). Поскольку эта информация идет в RFP, разрабатывать более детальные требования вам может потребоваться раньше, чем это бывает в проектах, разрабатываемых собственными силами (Morgan, 2009). Как минимум определите обширный набор пользовательских требований и нефункциональных требований для RFP. Когда начнется разработка по проекту, вам скорее всего потребуется указать все требования с бо́льшей точностью, чем это обычно делается при разработке системы собственными силами, особенно если сторонняя команда находится в другой стране. Если вы раньше проявляли излишнюю дотошность и многословность при детализации требований, то в аутсорсинговых проектах это будет как нельзя больше кстати. Именно на авторе требований лежит обязанность четко и ясно выразить ожидания заказчика. Если есть компоненты, создаваемые для сертификации или выполнения нормативных требований, обязательно включите их в RFP.

Как и при собственной разработке, визуальные модели требований повышают ценность функциональных и нефункциональных требований для аутсорсинговых команд. Создание множественных представлений требований расширяет диапазон коммуникации, поэтому желательно создавать больше моделей, чем при обычной разработке собственными силами. Такие представления, как визуальные модели, дополняющие письменные спецификации, особенно ценны при работе с командами разных культур и языков, потому что дает разработчикам что-то, с чем можно сравнивать свою интерпретацию требований. Но нужно быть уверенным, что разработчики понимают модели, которые вы им предоставляете. Если они не знакомы с нотацией моделей, они будут только дополнительным источником непонимания. Один менеджер разработки опасался, что письменных спецификаций требо-

ваний и моделей будет недостаточно для команды в другой стране, которая должна реализовать сложный пользовательский интерфейс (Beatty и Chen, 2012). Специально для этого аутсорсингового проекта была создана модель «отображение-действие-реакция», описанная в главе 19.

Прототипы также могут помочь уточнить ожидания команды подрядчика. Также, подрядчик может создать прототипы, чтобы показать заказчику свою интерпретацию требований и как подрядчик планирует действовать в соответствии с ними. Это один из способов организации более тесного взаимодействия между заказчиком и разработчиками, позволяющий как можно раньше корректировать ход проекта. Подробнее о создании и использовании прототипов см. главу 15.

Остерегайтесь неясных терминов, показанных в табл. 11-2, — они становятся причиной полной неразберихи. Как-то раз я читал спецификацию требований к ПО, предназначенную для сторонней организации-разработчика, в которой много раз повторялось слово «поддержка». Бизнес-аналитик, создавший эти спецификацию, признал, что в организации-подрядчике не понимали, как интерпретировать это слово в каждом случае. Словарь терминов особенно полезен при взаимодействии с теми, кто не владеет неявными знаниями тех, кто знаком со средой у заказчика. Для очень точного описания требований для разработки сторонней компанией можно использовать структурированный язык Planguage, см. главу 14 (Gilb, 2007).

Взаимодействие заказчика и подрядчика

Если с разработчиками нельзя организовать встречи «лицом к лицу», в режиме реального времени, придумайте другие механизмы, которые позволят вам быть в курсе дел по вашему проекту. В некоторых проектах разработки сторонними компаниями подрядчик помогает писать функциональные требования (Morgan, 2009). Это увеличивает стоимость проекта, но одновременно снижает риск неправильного понимания.

Надо рассчитывать на несколько циклов рецензирования требований. Для выполнения рецензирования в распределенной команде используйте средства совместной работы (Wiegers, 2002). Но нужно иметь в виду, что в некоторых культурах не принято критиковать работу других. Такие авторы могут воспринять указание на недостатки на свой личный счет (Van Veenendaal, 1999). В результате во время рецензирования рецензенты могут сидеть и ничего не говорить, чтобы не обидеть автора. Это вежливо и деликатно, но не способствует решению общей цели как можно быстрее обнаружить дефекты в требованиях, чтобы снизить стоимость и ускорить разработку. Выясните, нет ли таких культурных особенностей у ваших партнеров, чтобы установить реалистичные ожидания и стратегию рецензирования.

В график одного неудавшегося проекта, разрабатываемого сторонней организацией, был включен этап длиной в одну неделю под названием «Проведение семинаров по требованиям», следующим был указан этап реа-

лизации нескольких подсистем (Wiegers, 2003). Подрядчик забыл о важнейших промежуточных стадиях по документации, проверке и корректировке спецификации требований. Поскольку процесс разработки требований является итерационным и зависит от эффективного взаимодействия, необходимо предусмотреть время на эти циклы проверок. В том проекте заказчика и подрядчика разделял континент. При этом обмен информации при наличии огромной массы вопросов, возникающих при работе над спецификацией требований к ПО, был медленным. Неспособность своевременно решать вопросы, связанные с требованиями, нарушил график проекта, и со временем привел обе стороны в зал суда.

Рецензирование и прототипы позволяют понять, как подрядчик интерпретирует требования. Поэтапная разработка, при которой подрядчик периодически поставляет небольшие части продукта, считается одним из способов, обеспечивающих управление рисками. Он позволяет быстро корректировать курс, когда разработчики подрядчика из-за непонимания выбирают неверное направление. Если у подрядчика есть вопросы, задокументируйте их и включите ответы в требования (Gilb, 2007). Отслеживайте разрешение вопросов в средстве отслеживания дефектов, к которому есть доступ как у заказчика, так и подрядчика, как описано в главе 27.

У компаний, работающих по подряду над проектами самых разных типов, могут отсутствовать специальные знания предметной области или компании, которые критически важны для принятия правильных решений. Подумайте, может быть разумным предоставить определенное обучение сотрудникам подрядчика по проекту и прикладной предметной области до начала рецензирования, чтобы закрыть этот пробел в знаниях.

В аутсорсинговых проектах часто встречаются компании с разными культурами и отношением. Некоторые подрядчики так стремятся угодить, что соглашаются на то, что не смогут предоставить. Когда вы им укажете на ошибку, они будут пытаться сохранить лицо, не беря на себя всю ответственность за проблемы. Дополнительные культурные различия присутствуют в проектах оффшорной разработки. Некоторые разработчики не решаются обратиться за помощью или разъяснением. Они с неохотой говорят «нет» или «я не понимаю». Это может приводить к неправильному пониманию, неразрешенным дефектам и невыполнению обещаний. Чтобы избежать подобных проблем, используйте приемы выявления требований и организации работы, такие как чтение между строк, чтобы понять, что осталось невысказанным, и постановка открытых вопросов, чтобы четко понимать все проблемы и состояние проекта. Стоит согласовать базовые правила с членами команд — как местной, так и удаленной, и явно определить, как члены команды должны взаимодействовать друг с другом и совместно работать.

Разработчики, родной язык которых отличается от языка, на котором написаны требования, могут интерпретировать их буквально, не понимая нюансов или полностью не осознавая последствий. Они могут создавать неожиданные решения пользовательского интерфейса. Такие вещи, как форматы

дат, системы измерений (например, традиционная американская система единиц, единицы СИ или единицы британской системы мер и весов), значения цветов, а также порядок имени и фамилии, могут различаться в разных странах. Общаясь с людьми, родной язык которых отличается от вашего, выражайте свои намерения и желания максимально ясно и четко, используя простой язык. Избегайте использования разговорных выражений, жаргона, идиом, а также ссылок на явления культуры, которые могут понять неверно.

Одна оффшорная команда восприняла требования заказчика очень буквально. Дело выглядело так, как будто разработчики переводили каждое требование с английского на собственный язык, реализовывали его, после чего делали то же со следующим требованием, повторяя эту операцию, пока все требования не были реализованы. Формально предоставленный заказчику продукт отвечал этим требованиям, но совершенно не соответствовал ожиданиям. Разработчики не старались усложнить дело — они просто недостаточно хорошо понимали язык требований. Поэтому они так и не смогли ухватить суть того, что создают. Заказчику пришлось выполнить большую часть разработки самостоятельно, поэтому в результате ему пришлось заплатить вдвое больше, чем если бы с самого начала продукт создавался самостоятельно.

> **Внимание!** Не надо рассчитывать, что подрядчики будут интерпретировать неясные или неполные требования так же, как вы. Бремя передачи всей необходимой информации подрядчику лежит на заказчике; не отказывайтесь почаще беседовать с подрядчиком, чтобы снять все вопросы, касающиеся требований. Но на подрядчике лежит бремя активно задавать вопросы, не делая никаких предположений, которые могут оказаться неверными.

Управление изменениями

Вначале проекта установите взаимоприемлемый процесс управления изменениями, используемый всеми участниками независимо от того, где они находятся. Жизненно важно использовать единый набор веб-средств для работы с запросами на изменение и для отслеживания дефектов. У изменений всегда есть цена, поэтому управление изменениями, позволяющее контролировать границы проекта, жизненно важно при работе по контракту. Определите ответственных за принятие решений об изменениях и за механизмы коммуникации, обеспечивающие информацию всем нужным людям. В большинстве отдаваемых на сторону работ предусмотрены контрактные соглашения, в точности описывающие, что должна предоставить команда разработчиков. В контракте необходимо указать, кто платит за различные виды изменений, например за только что запрошенную функциональность или коррективы, вносимые в первоначальные требования. Трения, возникающие в случае расхождений между требованиями и предоставленным решением, регулируются договором. К несчастью, в таком случае проигрывают обе стороны (McConnell, 1997).

Критерии приемки

В соответствии с рекомендацией Стивена Кови «начинайте, думая об окончании» (Covey, 1989), определите, как вы будете оценивать приемлемость разрабатываемого по контракту продукта для вас и ваших клиентов. Как вы будете решать, когда пришло время окончательно рассчитываться с подрядчиком? Если критерии приемки оказываются удовлетворенными не полностью, кто отвечает за внесение корректив и кто платит за это? Включите критерии приемки непосредственно в RFP, чтобы подрядчик сразу понимал, чего ожидать. Утвердите требования прежде чем отдавать их аутсорсинговой команде, чтобы быть уверенным, что готовый продукт будет решать поставленную задачу. В главе 17 предлагаются некоторые подходы к определению критериев приемки, а также методы рецензирования и тестирования требований.

При надлежащем управлении поручение разработки сторонней компании может оказаться эффективным способом построения программной системы. Построение процессов взаимодействия с подрядчиком может оказаться непростой задачей по причине расстояния, языковых и культурных различий, а также возможного конфликта интересов. У поставщиков может отсутствовать мотивация устранять ошибки и неясные места в требованиях, выявленные в процессе выполнения проекта, если они будут получать деньги за решение проблем, возникающих после поставки или выпуска продукта. Для успеха аутсорсингового проекта жизненно важно иметь высококачественные, полные, ясные и прозрачные требования. Если требования, которые вы предоставили подрядчику, неполны или поняты неправильно, вина за провал проекта будет лежать в равной мере на вас, как и на подрядчике.

Глава 24

Проекты автоматизации бизнес-процессов

В организациях часто принимают решение полностью или частично автоматизировать ручные бизнес-процессы с помощью ПО — это позволяет сэкономить текущие расходы. На самом деле большинство ИТ-проектов предусматривают определенный объем автоматизации бизнес-процессов — в их числе наша система Chemical Tracking System и другие проекты, о которых мы говорили в этой книге. Процессы могут автоматизироваться путем разработки новой системы, доработки существующей системы или приобретением готового серийного решения. Если вы работаете над проектом автоматизации бизнес-процессов, нужно узнать о некоторых приемах работы с требованиями, прежде чем стараться совместить новые системы и обновленные бизнес-процессы.

Автоматизация бизнес-процессов встречается в подавляющем большинстве проектов по разработке ПО, поэтому в них находят применения многие приемы, описанные в других главах этой книги. В этой главе описывается схема, которая позволит справиться с подобными проектами, и указываются самые подходящие приемы, описанные в других частях книги. Здесь также представлены некоторые дополнительные приемы, которые не описаны в других местах книги.

Вот иллюстрация того, что иногда происходит в проектах автоматизации бизнес-процессов. У одного нашего клиента была таблица, которая применялась для оценки рисков при выдаче кредитов и в которой использовалась информация из более чем трехсот источников. Заинтересованные лица компании хотели получить программу, которая бы собирала входные данные и рассчитывала риск выдачи кредита, потому что на этот часто выполняемый процесс менеджеры по управлению рисками тратили очень много времени. Мы проанализировали, на какую часть этого процесса пользователи тратили больше всего времен, и быстро выяснили, что это сбор данных, которые нужно было вводить в электронную таблицу. Вычисления в электронной таблице выполнялись практически мгновенно. У команды разработчиков уже был доступ к большинству источников данных для заполнения электронной табли-

цы, поэтому доступная для реализации первая стадия проекта заключалась и извлечении и загрузке этих данных в таблицу. Бизнес-пользователям на какое-то время придется продолжить вручную собирать остальные входные данные. На второй стадии разработчики должны автоматизировать остальные источники данных. В команде решили, что не будут создавать программу для репликации вычислений в электронной таблице, потому что вычисления уже выполнялись достаточно быстро.

Этот пример иллюстрирует типичный проект автоматизации бизнес-процессов. В компании выделили занимающую много времени повторяющуюся операцию, которую решили ускорить с помощью соответствующего ПО. В процессе анализа выявляются узкие места и возможности повышения эффективности. Это вылилось в требования и проектные планы частичного решения, которое сэкономит компании значительное время, снизит затраты и сократит число ошибок ввода.

Моделирование бизнес-процессов

Выявление требований к автоматизации бизнес-процессов начинается с их моделирования. Определив, какие задачи пользователю требуется выполнять средствами системы, бизнес-аналитик может выработать необходимые функциональные требования, которые позволят пользователям выполнять эти задачи. Процессы, описывающие, как бизнес работает сейчас, называются *текущими процессами* (as-is processes). Процессы, описывающие то, как представляется работа бизнеса в будущем, называются *будущими процессами* (to-be processes).

Галерея акронимов, связанных с процессами

Существует много литературы по анализу бизнес-процессов (BPA), реинжинирингу бизнес-процессов (BPR), совершенствованию бизнес-процессов (BPI), управлению бизнес-процессами (BPM) и нотации и модели бизнес-процессов (BPMN). Эта глава не является исчерпывающим пособием по этим темам. В следующем списке приведены некоторые базовые определения этих концепций и их предназначение, но вы увидите, что они значительно перекрываются.

- **BPA (Business Process Analysis)** предусматривает понимание процессов в качестве основы для их совершенствования. Эта концепция похожа на моделирование процессов, описанное в Business Analysis Body of Knowledge («Свод правил по бизнес-анализу») (IIBA, 2009).
- **BPR (Business Process Reengineering)** состоит из анализа и перепроектирования бизнес-процессов с целью повышения их эффективности и продуктивности. BPR может ориентироваться на отдельные части процессов или предусматривать полный пересмотр процессов организации (Hammer и Champy, 2006).

- **BPI (Business Process Improvement)** предусматривает измерение и поиск возможностей для поэтапной оптимизации процесса (Harrington, 1991). В BPI-проектах часто используются концепция «Шесть сигм» и приемы бережливого управления (lean management practices) (Schonberger, 2008).
- **BPM (Business Process Management)** предусматривает определение всех бизнес-процессов компании, их анализ для повышения эффективности и производительности и совместная работа с компанией над изменением процессов (Harmon 2007; Sharp и McDermott, 2008). BPM может представлять собой определенное сочетание BPA, BPR и BPI.
- **BPMN (Business Process Model and Notation)** представляет собой систему условных графических обозначений для моделирования бизнес-процессов (OMG, 2011). BPMN может применяться в любой из описанных методик моделирования бизнес-процессов. Это полноценный язык условных обозначений, который можно применять, когда не хватает базового синтаксиса swimlane-диаграмм.

При реализации BPA, BPR, BPI и BPM применяются самые разнообразные методы и инструменты, которые нужны ваш проект предусматривает серьезное изменение бизнес-процессов. Все четыре концепции являются признанными методиками для понимания бизнес-проблем и возможностей. Описанные в этой книге приемы разработки требований приобретают смысл, когда компания решает, что программный компонент должен стать частью решения, призванного оптимизировать бизнес-процессы.

Использование текущих процессов для вывода требований

Описанные далее шаги помогут вам смоделировать ряд бизнес-процессов и выявить требования к приложению, автоматизирующему все или часть этих процессов. Последовательность шагов не всегда одинакова, и не в каждом проекте нужны они все. В некоторых случаях будущие потоки процессов могут изучаться раньше в качестве способа анализа расхождений или для обеспечения, чтобы новая система была лучше старой и не представляла собой всего лишь старую систему в новой упаковке. В целом, последовательность такова:

1. Как всегда при разработке ПО нужно начинать с понимания бизнес-целей, чтобы можно было привязать каждую к одному или нескольким процессам.
2. Используйте структурные схемы организации для выявления всех задействованных организаций и возможных классов пользователей будущего программного решения.
3. Определите все необходимые бизнес-процессы, предусматривающие участие этих классов пользователей.

4. Задокументируйте все существующие бизнес-процессы, используя диаграммы потоков, диаграммы действий или swimlane-диаграммы. Любая из этих трех моделей является практичным вариантом представления задач пользователей. Пользователи могут быстро прочитать их и указать неправильные шаги, роли или логику принятия решений (Beatty и Chen, 2012). Вам нужно самому принять решение об уровне детализации моделирования существующих процессов, чтобы получить достаточно информации для выполнения следующих шагов этого списка.
5. Проанализируйте существующие процессы для определения самых больших возможностей оптимизации за счет использования автоматизации. Если это неочевидно, нужно собрать информацию о том, сколько времени занимает выполнение определенных шагов или целых процессов. Это моделирование можно выполнить с помощью модели ключевых индикаторов производительности (key performance indicator model, KPIM), описанной далее в этой главе. Этот шаг помогает выявить возможности и, если программное решение окажется уместным, задать границы той части проекта, которая связана с разработкой ПО. Обязательно убедитесь, что устраняются настоящие узкие места процесса, ускорение в которых обеспечит сокращение времени всего процесса.
6. Для каждого процесса, подпадающего под автоматизацию, пройдитесь по процессу с соответствующими заинтересованными лицами, чтобы выявить требования, обеспечивающие каждый этап потока. На этом этапе будут полезными приемы, описанные в главе 7. Если это уместно, может быть полезным выяснить отраслевые стандарты для моделируемого процесса, чтобы поставить цели по оптимизации.
7. Отследите связь требований с конкретными этапами потока процесса, чтобы понять, не упустили ли вы какие-либо требования для отдельных этапов. Если есть этапы процесса, которым не соответствуют требования, убедитесь, что эти этапы в рамках проекта не автоматизируются.
8. Задокументируйте потоки будущих процессов, чтобы в компании могли подготовиться к новой системе и определить пробелы, которые при наличии новой системы могут остаться в процессе. Можно также создать варианты использования, предоставляющие более подробную информацию о том, как пользователи будут взаимодействовать с новой системой. Эта информация позволяет разработчикам быть уверенными, что они создают систему, отвечающую ожиданиями компании, и помогает пользователям понять, какую пользу они от нее получат. Потоки будущих процессов и варианты использования можно использовать для разработки обучающих материалов и определения других переходных требований. Это позволяет заинтересованным лицам понять не только, с чем им предстоит иметь дело, но и какие ручные действия и автоматизированные системы будут устранены.

Когда ПО не нужно

Иногда для улучшения бизнес-процессов не нужно ничего автоматизировать. У одной компании был внутренний веб-сайт, где хранились имена людей, работавших над определенными клиентскими проектами: специалист по продажам, консультанты по внедрению и т. п. Информация о специалистах по продажам была всегда точной, а информация о консультантах по внедрению не соответствовала действительности более чем в половине случаев. Это приводило к тому, что людям приходилось искать нужного им человека. Если умножить две-три минуты, которые тратит на это каждый из 200 сотрудников отдела по крайней мере раз в неделю на протяжении года, на стоимость их времени, получится очень немаленькая сумма. Формулировка проблемы: отсутствует процесс между отделами продаж и внедрения по обновлению информации о проекте внедрения после его начала. Решение: определить человека, который будет отвечать за сбор и ручное обновление контактной информации о сотрудниках отдела внедрения по каждому клиенту. В таком процессе никакое новое ПО не решило бы проблему.

Проектирование в первую очередь будущих процессов

С информационными системами и бизнес-процессами такая же ситуация как и с вопросом о первенстве курицы или яйца. В некоторых случаях люди ожидают, что создание новой системы поспособствует улучшениям или изменениям в процессах. Однако на практике способ использования приложения не обеспечивает требуемых изменений в бизнес-процессах. Изменения процессов предусматривают перемены в культуре и обучение пользователей, а этого программная система обеспечить не в состоянии. Некоторые клиенты считают, что ответственность за успешное развертывание и руководство внедрением соответствующих бизнес-процессов лежит на команде разработки. Пользователи не примут новую систему просто потому, что так им сказал разработчик.

Во многих случаях лучше разработать новые бизнес-процессы и лишь после этого приступать к оценке изменений, которые нужно внести в архитектуру информационных систем. Для надлежащей поддержки нового бизнес-процесса может потребоваться внести изменения во многие системы. Анализ того, какие пользователи будут работать с системой и как они будут применять ее для выполнения своей работы позволит вам выявить правильные пользовательские требования, что, в свою очередь, будет способствовать положительному восприятию системы пользователями. Параллельная разработка новых процессов и новых приложений позволяет обеспечить хорошую координацию между ними.

Моделирование метрик бизнес-производительности

Важно понимать, какие метрики бизнес-производительности важно учитывать при автоматизации бизнес-процессов. Перед началом работы над документом концепции и границ проекта уже могут быть определены метрики успеха (см. раздел «1.4 Критерии успеха» в главе 5). Если это не так, разработанные здесь метрики бизнес-производительности помогут завершить документ концепции и границ проекта. В приведенном ранее в этой главе примере вас может интересовать, сколько времени занимает ручное заполнение электронной таблицы и как быстро это должно происходить в автоматизированном решении.

Индикаторы KPIM увязывают бизнес-процессы с соответствующими им важными метриками производительности. Индикаторы KPIM отображаются как блок-схемы, swimlane-диаграммы или диаграммы действий с указанием ключевых показателей производительности (Key Performance Indicators, KPI) на отдельных шагах. На рис. 24-1 показан пример KPIM (в виде блок-схемы) для проекта автоматизации вычислений в электронной таблице.

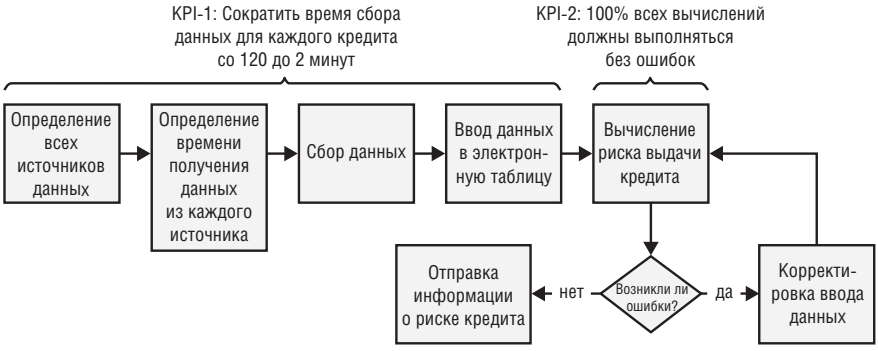


Рис. 24-1. Пример KPIM для процесса вычисления риска выдачи кредита

Самые важные процессы для автоматизации — те, в которых нужно сохранить или улучшить самые важные метрики. Определите текущее базовое значение для каждой метрики, чтобы по ходу автоматизации процесса можно было определить, улучшаются ли они, как требовалось. Имейте в виду, что для улучшения одних метрик бизнес-производительности может потребоваться ухудшить другие метрики. Подробнее о компромиссах между разными атрибутами качества см. главу 14. Здесь применяется аналогичный подход, но в данном случае компромисс заключается в предпочтении одних метрик производительности в ущерб другим, возможно в других частях компании. Отслеживание связей требований с шагами потока процессов, которые также можно увязать с KPI-показателями, позволяет определить приоритеты запланированных к реализации требований.

Вам может потребоваться встроить в систему функциональность периодической проверки соответствующих KPI-индикаторов для оценки эффек-

тивности нового решения автоматизации и уведомления, когда показания индикаторов падают ниже допустимых. В примере с электронной таблицей система может измерять, сколько времени занимает сбор всех исходных данных для определения, укладывается ли система в двухминутный норматив. Если нет, могут потребоваться дополнительные изменения.

Бизнес-пользователи часто полагают, что автоматизация ручных операций это всегда хорошо. Но все проекты разработки стоят денег. Бизнес-аналитики помогут вам определить, какие процессы стоит автоматизировать, а какие нет. Например, в Seilevel (компания Джой, автора этой книги) используют серийные решения для управления процессом продаж и выделением людских ресурсов. Мы создаем отчет с помощью программы управления продажами и вручную вводим будущие проекты в средство выделения людских ресурсов для прогнозирования потребностей в ресурсах. Наш менеджер должен делать это по крайней мере раз в неделю. Это занимает у него примерно полчаса — сгенерировать отчет о процессе продаж, принять решение, какие проекты нужно перенести из продаж, когда начнется выполнение этих проектов и сколько для них потребуется ресурсов. Мы прикинули, нужно ли нам включать функцию интеграции для автоматического переноса данных из одной системы в другую. Интеграция выполняется просто — надо только включить соответствую функцию, но она потребует дополнительной разработки для автоматизации процесса принятия решения, который выполняет наш менеджер. Разработка спецификации и автоматизация логики потребует слишком много усилий, которые себя не оправдают.

Приемы, рекомендуемые к использованию в проектах автоматизации бизнес-процессов

Многие приемы, описанные в остальных частях этой книги важны для проектов автоматизации бизнес-процессов. В табл. 24-1 перечислены самые важные приемы, описано, как они применяются в таких проектах, и указано, в каких главах искать соответствующую информацию.

Табл. 24-1. Информация об описанных в других главах приемах, которые будут полезны в проектах автоматизации бизнес-процессов

Прием	Глава
Определение классов пользователей, в которых могут быть процессы, нуждающиеся в автоматизации	Глава 6
Создание и расширение моделей данных для информации, которая обрабатывается вручную	Глава 13
Создание матрицы ролей и разрешений для выявления требований к безопасности, которые ранее реализовывались вручную	Глава 9

Табл. 24-1. (*окончание*)

Прием	Глава
Определение бизнес-правил, которые нужно автоматизировать, если будут автоматизированы зависящие от них процессы	Глава 9
Создание блок-схем, swimlane-диаграмм или вариантов использования, чтобы показать, как пользователи выполняют свои задачи сейчас и как будут выполнять их после автоматизации	Главы 8 и 12
Использование диаграмм потоков данных (Data Flow Diagram, DFD) для определения процессов, которые можно было бы автоматизировать и создание новых DFD, чтобы показать, как новые автоматизированные процессы будут взаимодействовать с существующими частями системы	Глава 12
Адаптация бизнес-процессов, чтобы обеспечить возможность использования серийного решения	Глава 22
Создание матриц связей для сопоставления шагов процесса и требований	Глава 29

Вы скорее всего будете применять описанные в этой главе концепции и приемы практически во всех проектах информационных систем, над которыми будете работать. Когда нужно автоматизировать часть или весь бизнес-процесс, используйте описанный в этой главе каркас, чтобы гарантировать, что вы полностью понимаете цели автоматизации процесса и сопутствующие требования. Это позволит всем понять ожидания пользователей, а разработчики смогут создать успешное решение, которое обеспечит ожидаемые бизнес-преимущества.

Глава 25

Проекты бизнес-аналитики

Большинство нормальных людей не смотрят на данные просто для развлечения — они изучают представления данных, принимая решения о том, что делать, будь то решение о действии или о бездействии. В некоторых случаях программные системы автоматизируют процесс принятия решений путем интерпретации данных и выполнения операций по заданным алгоритмам и правилам. Основная цель проектов *бизнес-аналитики* (business analytics) [в англоязычной литературе ее еще называют business intelligence или business reporting] — разработка систем, которые превращают объемные и часто очень сложные наборы данных в осмысленную информацию, пригодную для принятия решений. Во многих других видах проектов присутствуют элементы бизнес-аналитики, представленные в этой главе принципы также применимы к таким проектам.

Решения, принимаемые с применением систем бизнес-аналитики, делятся на стратегические, оперативные и тактические. Топ-менеджер может изучать информационную панель с данными о продуктивности команды продаж, чтобы принять решение, кому дать повышение (тактическое решение), какие продукты требуют других стратегий маркетинга (оперативное решение) или какие продукты на каких рынках размещать (стратегическое решение). Вообще говоря, все программные системы, содержащие компонент аналитики, должны позволять пользователям принимать решения по улучшению организационной производительности в том или ином измерении.

На рынке представлено много коммерческих приложений бизнес-аналитики. Бизнес-аналитику, планирующему использовать одно из этих приложений, скорее всего потребуется выполнить сбор требований для выбора и реализации, используя описанный в главе 22 процесс.

Цель этой главы — служить введением в вопросы, которые нужно рассмотреть при разработке требований к ПО для проектов бизнес-аналитики. Берт Брис много написал о выполнении бизнес-анализа в подобных проектах (Brijs, 2013). Он предоставил много определений базовых принципов, примеров в конкретных предметных областях, вопросов, требующих ответов, и проблем, с которыми можно столкнуться.

Общие сведения о проектах бизнес-аналитики

В большинстве информационных систем на отчеты приходится лишь небольшая часть реализованной функциональности. Однако в проектах бизнес-аналитики сложные отчеты и возможности манипулирования их содержанием являются основной функциональностью. Часто результаты анализа встраиваются в приложения, автоматизирующие процесс принятия решений. У проектов бизнес-аналитики много уровней, и каждый нуждается в определении отдельных требований к ПО. В этих проектах нужно понимать, какие нужны данные, какие операции с ними выполняются, а также как форматируется и распределяется готовая к использованию информация (рис. 25-1). Последовательность этих действий может быть произвольной. Пользователь может сначала работать с данными, после этого выяснить, что нужен другой их анализ или даже другие источники данных.

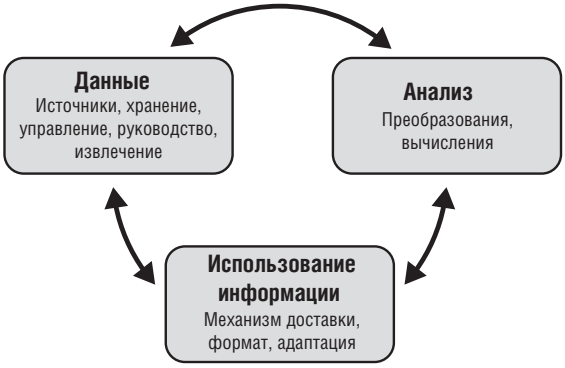


Рис. 25-1. Компоненты простого каркаса бизнес-аналитики

В прошлом организации разворачивали аналитические проекты, в основном ориентированные на то, что Международный институт аналитики называет «описательной аналитикой» (International Institute for Analytics, 2013). Сюда входят отчеты, говорящие заинтересованным лицам, что происходит или произошло в организации. Новейшие тенденции демонстрируют смещение акцентов организаций на «аналитику прогнозирования». Пользователи организуют, манипулируют и анализируют информацию, стремясь спрогнозировать, что может произойти в будущем, в отличие от интерпретации прошлых событий. На рис. 25-2 показано место различных аналитических приложений на шкале от описательности до прогнозирования.

Если в организации решат начать аналитические проекты, бизнес-аналитикам будет поручено выявление и спецификация требований для этих проектов, скорее всего без понимания, с чего начать. Стратегические возможности, новые аналитические технологи и быстрый рост объема собираемых данных может отпугивать. Конечный результат разработки требований для проекта бизнес-аналитики будет похож на обычные результаты: набор бизнес-требований, а также пользовательских, функциональных и нефунк-

циональных требований. Однако многие описанные в этой книге приемы выявления требований недостаточны для выявления и спецификации требований в проектах этого типа. Потоки процессов, варианты использования и пользовательские истории могут показать, что кому-то нужно генерировать аналитические результаты, а требования к производительности описывают, как быстро нужно их получать, но все это ничего не говорит о сложном знании, необходимом для реализации системы.

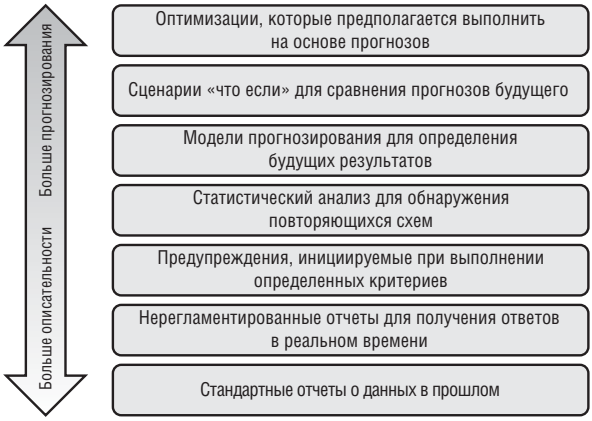


Рис. 25-2. Шкала различных типов аналитики (Patel и Taylor, 2010; Davenport, 2013)

Если организация новичок в аналитике, нужно выполнить ряд небольших пилотных проектов, чтобы продемонстрировать ценность аналитики и научиться на собственном опыте (Grochow, 2012). Аналитические проекты хорошие кандидаты для поэтапной накопительной разработки, если команда может определить самые важные и критичные по времени решения, которые нужно реализовать в следующей итерации разработки.

Еще одна причина применения поэтапной разработки заключается в том, что у заинтересованных лиц компании бывают сложности с формулировкой бизнес-задач, которые им нужно решить за счет аналитического проекта, особенно если это их первый такой проект. У некоторых заинтересованных лиц может не хватать опыта стратегического мышления. Для других может быть сложным увидеть возможности, предлагаемые аналитическими технологиями, помимо знакомых таблиц. Пользователям могут так понравиться аналитические возможности, что они начнут заваливать команду разработчиков заказами новых, кажущихся ценными функций. Выявление требований может начаться с небольшого курса о том, какие новые возможности может предоставить решение бизнес-аналитики помимо традиционных средств отчетности (Imhoff, 2005). Разработка решения аналитики небольшими порциями может дать пользователям возможность изучить базовые возможности и прояснить для себя, что им на самом деле нужно.

Разработка требований в проектах бизнес-аналитики

Как и в других проектах разработки ПО, в проектах бизнес-аналитики нужно первым делом установить бизнес-цели и определить их приоритеты в рамках работы. Если заинтересованные лица заказывают аналитический проект, они уже решили, что это будет решение, и не обязательно тщательно продумали свои цели. В процессе изучения базовых бизнес-целей может оказаться, что бизнес-аналитика совсем не подходящее решение. Чтобы помочь заинтересованным лица сформулировать свои настоящие бизнес-цели, можно задать такие вопросы:

- Почему вы думаете, что аналитическое решение поможет вам получить желаемые бизнес-результаты?
- Чего вы хотите добиться за счет реализации аналитической отчетности?
- Как вы ожидаете использовать аналитику для улучшения своих бизнес-результатов?
- Как вы надеетесь использовать улучшенные возможности создания отчетов и результаты прогнозирования?

Эффективная дальнейшая стратегия выявления требований заключается в определении спецификации требований на основе решений, которые нужно принимать заинтересованным лицам для достижения своих бизнес-целей. Попробуйте применить следующий прием анализа (Taylor, 2013):

1. Опишите бизнес-решения, которые будут приниматься на основе выдаваемой системой информации.
2. Свяжите эти решения с бизнес-целями проекта.
3. Выполните декомпозицию решений, чтобы обнаружить вопросы, на которые нужно ответить, иерархию предшествующих вопросов, на которые нужно ответить, чтобы получить ответ на вопросы более высокого уровня, а также чтобы определить, какую роль аналитическая информация будет играть в получении ответов на эти вопросы.
4. Определите, как аналитика может помочь в принятии этих решений.

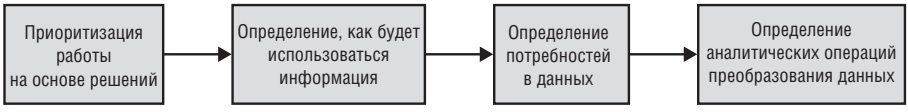


Рис. 25-3. Процесс определения требований в проектах бизнес-аналитики

На рис. 25-3 показан подход выявления и спецификации требований в аналитических проектах. Нужно определить пользовательские требования, чтобы описать, как будет использоваться аналитическая информация и какие решения будут приниматься на ее основе. Понимание ожидаемых способов использования позволяет определить, как полученную информацию распределять среди конечных пользователей и какую информацию они должны

видеть. Это знание, в свою очередь, позволяет определить требования к самим данным и анализу, который должен выполняться. В оставшейся части детально описывается каждый из этих этапов.

Приоритизация работы на основе решений

В проектах большинства типов приоритеты функций можно определять, ориентируясь, на то, что они вносят в достижение бизнес-целей. То же верно в аналитических проектах, за исключением того что в них нет отдельных «функций», которые можно приоритизировать. Вместо этого бизнес-цели используются для определения приоритетов бизнес-решений, обслуживаемых решением, основываясь на их вкладе в достижение этих целей. Например, принятие решений о том, какие продукты продавать, будет сильнее влиять на увеличение доходов, чем решения о времени отпусков сотрудников отдела продаж. Поэтому скорее всего в первую очередь будет реализовываться аналитика и отчеты для определения продуктов, продаваемых в первую очередь.

Решения должны формулироваться так же недвузначно, как и требования. Вот пример хорошей формулировки решения: «Вице-президенту по маркетингу нужно каждый квартал решать, какой бюджет на маркетинг выделить отдельным регионам, основываясь на текущих и запланированных продажах по регионам». Как и при выявлении требований в других проектах по разработке ПО, важно понять базовые потребности заинтересованных лиц, не зацикливаясь на представленном решении. Если заинтересованные лица просят предоставить определенные данные или отчеты, задайте вопрос: «Зачем вам нужна эта информация?» или: «Как получатель будет использовать этот отчет?» Затем пройдитесь в обратном порядке, чтобы определить их решения и цели.

Существуют приемы управления решениями, которые помогают заинтересованным лицам определить решения, которые они могут или должны принимать (Taylor, 2012). Модель решений, связывающая информацию (данные) и знание (политики или правила, отграничивающие решения) с соответствующими решениями могут помочь организовать решения для расстановки приоритетов (Taylor, 2013).

Определение, как будет использоваться информация

Результаты сложной аналитики должны предоставляться в удобной форме заинтересованным лицам или системам, действующим на основе этой информации. Бизнес-аналитик должен также определить, насколько «интеллектуальной» должна быть система, то есть какая часть решения лежит на человеке и какая автоматизируется системой. Это различие будет определять тип вопросов, задаваемых бизнес-аналитиком при выявлении требований.

В одной организации руководитель отдела продаж хотел каждое утро видеть в панели мониторинга множественные представления данных. Этот отчет должен был включать продажи за предыдущий день с разбивкой по

семействам продуктов, общий объем продаж в сравнении с продажами конкурентов и объемы продаж по ценовым диапазонам. Им нужно было десять различных фильтров (в частности, по временным рамкам, изменениям и регионам), которые можно было бы менять и моментально видеть изменения в отчетах. Например, обнаружив проблему в одном ценовом диапазоне, пользователь мог бы изменить фильтр, чтобы увидеть более подробное представление цены с разбивкой по регионам. После этого он мог бы опуститься на еще один уровень детализации, где показываются ценовой диапазон по регионам и семействам продуктов. Подобная гибкость — стандартная функциональность, которая должна присутствовать в системах бизнес-аналитики.

Использование информации людьми

Поняв, какие решения пользователям нужно принимать на основе результатов работы аналитической системы, можно определить оптимальные пути предоставления им этой информации. Бизнес-аналитик должен учесть следующие три аспекта доставки информации:

- **Механизм доставки** Как информация физически предоставляется конечному пользователю? Какие средства пользователь задействует для ее просмотра: приложения электронной почты, порталы, мобильные устройства или что-то другое?
- **Формат** В каком формате предоставляется информация: отчеты, панели мониторинга, «сырые» данные и т. п.?
- **Гибкость** Насколько широки возможности пользователя манипулировать полученной информацией?

Спектр предоставляемой информации простирается от предоставления каждому пользователю персонального представления данных (локальной копии электронной таблицы) до распределения централизованных агрегированных данных пользователям (распространяемые по электронной почте таблицы со стандартным представлением панели мониторинга) и до предоставления пользователям данных с возможностью самостоятельной работы с ними (портал, позволяющий создавать нерегламентированные запросы).

Как и с требованиями к другим типам программных систем, использование информации в аналитическом проекте чаще всего фиксируется в форме пользовательских требований и спецификаций отчетов. Описанные в других частях книги приемы, такие как потоки процессов, варианты использования и пользовательские истории, применяются для определения, как пользователи планируют работать с информацией в своей повседневной работе. Но не надо зацикливаться на определении полей данных в отчетах — используйте решения, которые требуется принять, чтобы определить, как пользователи должны получать результаты аналитики, как это должно выглядеть и какие у них должны быть возможности манипуляции данными.

Описанные в главе 13 таблицы отчетов бывают полезными в большинстве аналитических проектов. Может потребоваться расширить эти модели для

поддержки более сложного анализа за счет применения уровней в спецификации отчетов (Beatty и Chen, 2012). Пользователи аналитических данных часто желают видеть информацию в виде панели мониторинга с множественными диаграммами и отчетами на одной странице. Раздел «Панели мониторинга» в главе 13 поможет определить такие требования к панелям мониторинга. Некоторые отчеты предоставляют пользователям возможность определенным образом работать с представлением отчета, например использовать фильтры (Franks, 2012). Описанная в главе 19 модель «отображение-действие-реакция» представляет ценность для определения более сложных требований для работы с данными в отчетах, когда простой табличной структуры отчета недостаточно. Эти модели фиксируют сложные элементы пользовательского интерфейса в отчетах, такие как фильтры или изменения отображения при развертывании.

Помимо пользовательских требований и требований к отчетам, понимание того, как используется информация, может обнаружить новые процессы и требования к безопасности, нуждающиеся в описании. Вот пример: президент небольшой компании еженедельно получает отчет о прибылях и расходах. Если все выглядит правильно, он делится им с топ-менеджерами, но только с ними, а это подразумевает потребность в управлении доступом. Описанные в главе 14 требования по безопасности могут также требоваться для атрибутов данных, представлений отчетов и доступа к порталу. Например, вице-президентам по продажам в регионах может быть доступна информация только о продажах в своих регионах, а вице-президент по глобальным продажам может просматривать сведения по всей компании. Подобные требования к атрибутам качества применимы к проектам бизнес-аналитики, как и к любым другим программным проектам.

Использование информации в системах

Важно заметить, что информация аналитических проектов может использоваться непосредственно программными системами без доставки ее живым пользователям. Аналитика может встраиваться в приложение как одна из повседневных операций. Например, магазины розничной торговли могут использовать историю покупок клиента для определения, на какие товары предоставлять ему персональные скидки с расчетом, что он будет больше приобретать таких товаров. В одной сети розничной торговли узнали о беременности одного из авторов этой книги (речь идет о Джой) всего лишь через месяц после того, как сама автор узнала об этом, и стали присылать по электронной почте рекламу детских товаров. Другие примеры — система, печатающая купоны для покупателя в бакалейном магазине, основываясь на его текущих или предыдущих покупках, целевая реклама для посетителей веб-сайта, а также приложения центра вызовов, определяющая, что можно предложить позвонившему в данный момент клиенту.

В таких ситуациях механизм доставки и формат информации может определяться посредством требований к внешним интерфейсам. Однако по-

прежнему важно понимать, как будет использоваться эта информация, чтобы правильные данные преобразовывались нужным образом и поставлялись в другую систему в удобной для использования форме.

Определение потребностей в данных

В основе всех решений бизнес-аналитики лежат данные. В таких проектах для разработки и поддержки решений для работы с данными многие организации привлекают специалистов по данным. Бизнес-аналитики могут определить требования к источникам данных и механизмам хранения, управления и извлечения данных, но они вправе также привлекать для помощи специалистов по данным на ранних этапах разработки требований. Бизнес-аналитики могут помочь в определении типов данных, которые нужно собирать и анализировать, общего объема данных, с которыми будет иметь дело, и сколько данных будет накапливаться со временем. Однако специалисты по данным будут лучше знать, какие данные доступны, где они находятся, какие возможны сложности и как данные лучше использовать.

Так как аналитические проекты часто ориентированы на обнаружение новых стратегий для компаний, эти проекты могут предусматривать определение новых источников данных для анализа. Важно полностью понимать требования к данным, чтобы технические команды могли спроектировать инфраструктуры, которые в аналитических проектах бывают очень сложными. Например, для достижения поставленных перед проектом целей архитекторы могут полностью перепроектировать существующее решение хранения данных.

Большие данные

Термин *большие данные* (big data) обычно описывают массив данных, отличительные особенности которых — большой объем (много данных), высокая скорость (данные быстро поступают в организацию) и/или высокая сложность (данные очень разнородны) (Franks, 2012). Управление большими данными предусматривает обнаружение, сбор, хранение и обработку больших объемов данных быстро и эффективно. Джилл Дайк дает сводное определение, что представляют собой большие данные с точки зрения управления (Dyche, 2012).

Чтобы осознать понятие больших данных, подумайте о вашем личном взаимодействии с данными за один день: социальные сети, сообщения электронной почты, цифровые фотографии и электронные транзакции. Стоит сказать, что за получасовой полет серийный самолет генерирует 10 *терабайт* данных (Scalable Systems, 2008). Природа функционирования компаний сейчас такова, что объемы доступных им данных увеличиваются взрывообразно. Поэтому все важнее иметь приложения, которые помогают пользователям извлекать ценные знания из огромных массивов данных.

Описанные в главе 13 модели данных лучше всего подходят для представления реляционных хранилищ данных. Если между объектами данных есть какая-то логическая связь, бизнес-аналитик может смоделировать с помо-

щью диаграммы «сущность–связь». Если атрибуты данных известны и следуют определенному соглашению, будут также полезными словари данных. К сожалению, большие данные часто структурированы частично или вообще неструктурированы.

Неструктурированные данные, например голосовую почту и текстовые сообщения, нельзя представить в виде традиционных строк и столбцов. Проблема с неструктурированными данными в том, что совершенно непонятно, как и где начинать поиск нужной информации (Davenport, Harris и Morrison, 2010). Например, приложение в государственном агентстве по борьбе с терроризмом может сканировать интернет-трафик на предмет обнаружения определенных слов, например «бомба», но программа должна анализировать слово в контексте, чтобы понимать его смысл. Слово «бомба» может быть признаком террористической угрозы, содержаться в статье о роли авиации во Второй мировой войне или описывать песню, внезапно приобретшую популярность.

Хорошие новости заключаются в том, что у большинства данных есть какая-то структура в виде метаданных (Franks, 2012). К частично структурированным источникам данных можно отнести сообщения электронной почты, файлы изображений и видео. Поскольку у частично структурированных данных есть метаданные, предоставляющие какую-то информацию о структуре и содержимом данных, бывает возможным создать диаграммы «сущность–связь» и словари данных для представления того, что известно о данных.

Требования, основанные на данных (но не в БД)

Многие требования к данным, которые нужно определять в аналитических проектах, похожи на требования в других проектах информационных систем. Хотя природа этих требований может отличаться, вопросы, задаваемые в процессе выявления требований, часто совпадают. Не забывайте, что большинство больших данных генерируются автоматизированными системами и обычно представляют для организации новый вид источника данных, а это означает, что на определение требований к данным потребуется затратить больше усилий (Franks, 2012). Многие требования к данным можно вывести из критериев управления принятием решений, полученных от соответствующих заинтересованных лиц. Например, для решений, которые приходится принимать ежечасно, нужны одни данные, а для тех, что принимаются раз в квартал, — другие. Они могут отличаться тем, как часто обновляются исходные данные, когда данные извлекаются из источника и как долго они хранятся.

Брис создал контрольный список ожиданий заинтересованных лиц в отношении бизнес-аналитики и типов вопросов, позволяющих выявить эти ожидания (Brijs, 2013). Вот примеры вопросов, которые бизнес-аналитик может задавать для выявления требований, связанных с данными.

Источники данных

- Какие объекты и/или атрибуты данных вам нужны? Из каких источников вы будете получать эти данные?

- Если ли у вас уже доступ ко всем этим источникам данных? Если нет, то где есть нужные данные? Нужно ли разработать требования для наполнения этих источников нужными данными?
- Какие внешние и внутренние системы предоставляют данные?
- Насколько вероятно, что эти источники будут меняться со временем?
- Требуется ли начальный перенос исторических данных из старого хранилища в новое?

Хранение данных

- Какой объем данных на текущий момент?
- Какой ожидается рост объема данных и за какой период времени?
- Какие типы данных нужно хранить?
- Как долго нужно хранить данные? Какой уровень безопасности хранения данных нужно обеспечить?

Управление и данными

- Каковы структурные характеристики данных?
- Какие изменения в структуре и значениях данных будут происходить со временем?
- Какие предварительные преобразования данных необходимо выполнить перед сохранением или анализом данных?
- Какие преобразования необходимы для приведения данных из разнородных источников к единому формату?
- При каких условиях можно удалять старые данные? Нужно ли архивировать или удалять старые данные?
- Какие требования к целостности призваны защитить данные от неправомерного доступа, потери или повреждения?

Извлечение данных

- Как быстро пользователи ожидают получить результаты выполнения запросов?
- Как нужно извлекать данные — в реальном времени и в пакетном режиме? Если не в реальном времени, то какова должна быть частота пакетной обработки?

Как и с другими требованиями, надо обеспечить, чтобы связанные с данными требования не ограничивали возможности разработчиков определенным дизайном.

Определение аналитических операций преобразования данных

Анализ — это вычислительное ядро описанных в этой главе описываемых в этой главе проектов, которое преобразует данные в ответы на поставленные вопросы (Franks, 2012). Пользователь определяет вопрос, получает данные, которые, как он надеется, содержат ответ, анализирует данные в поиске от-

вета и принимает решение по вопросу. Или данные может анализировать система для получения ответа и выполнения соответствующего действия.

Все этого замечательно, если вы знаете, что ищете. Однако один из проблемных аспектов многих проектов бизнес-аналитики состоит в том, что человек, ответственный за принятие решения, не всегда знает, что он должен увидеть в данных. Ему могут требоваться наличие в аналитических инструментах определенных объектов и атрибутов данных, которые можно исследовать, выполнять разные запросы для получения ответов на вопросы типа «что, если». Он буквально не знает, что он не знает, но надеется в процессе изучения данных обнаружить что-то полезное и пригодное для принятия решения. Вот почему важно начать с понимания, какие решения ответственные за принятие решений заинтересованные лица пытаются принять. Даже если заинтересованное лицо не знает точно, что ищет, все равно оно должно уметь определить тип задачи, которую решает. Определение необходимого анализа данных подразумевает более масштабное мышление (Davenport, Harris и Morrison, 2010). Бизнес-аналитик с хорошими творческими навыками может в процессе работы с заинтересованными лицами определить новые идеи, которые можно изучать с применением результатов анализа.

Как показано на рис. 25-2, результаты аналитики предоставляют возможности принятия решений, попадающие в диапазон от описательности до прогнозирования. Для выявления требований к анализу данных можно задать вопросы, подобные перечисленным ниже (Davenport, Harris и Morrison, 2010).

- Какой временной диапазон будет анализироваться: прошлое, настоящее или будущее?
- Если это прошлое, то какие откровения вы хотите в нем увидеть?
- Если это настоящее, то что вам нужно узнать о текущей ситуации, чтобы немедленно предпринять какие-то действия?
- Если это будущее, то какие прогнозы или решения вы хотите получить или принять?

Эти вопросы помогут определить функциональные требования, определяющие виды анализа, который должна выполнять система. Так как аналитика является новинкой для многих организаций, может потребоваться выполнить определенное исследование, чтобы выяснить, как другие организации используют аналогичные данные для улучшения процесса принятия решений. У бизнес-аналитика есть возможность, и даже обязанность, помочь заинтересованным лицам понять, как находить применение аналитике там, где они себе и не представляли.

В некоторых видах анализа требуется применение сложных алгоритмов обработки, фильтрации и организации данных (Patel и Taylor, 2010). Представьте, что в розничном магазине хотят, чтобы при входе клиента в магазин воспроизводилось адресный видеоролик. Камера может сканировать посетителя, а программа распознавания — определять некоторые его характеристики (пол, возраст, одежду, направление взгляда), после чего встроен-

ная в систему логика сможет принять решение, какой рекламный видеоролик воспроизвести. Подобная логика принятия решений часто представлена в виде таблиц и деревьев решений, описанных в главе 12.

Важно понимать последствия автоматизированного принятия решений и явно выражать требуемое поведение системы принятия решений. Есть один поучительный пример систем, отслеживающих социальные сети для принятия решений об операциях с акциями. В 2013 году в новостном канале социальных сетей проскочило сообщение о том, что президент США был ранен при взрыве. Встроенные в определенные автоматизированные системы алгоритмы запустили продажи акций, что инициировало продаж акций другими системами, обнаружившими снижение показателей рынка, и все это произошло в считанные секунды после публикации новостей. К счастью ошибка была быстро обнаружена, и оперативные действия людей развернули резкое падение рынка, вызванное автоматическими торговыми системами. Может, системы вели себя в точности, как ожидалось, но возможно, что в них отсутствовала логика принятия решений, ограничивающая влияние ошибок.

Одна из самых ценных особенностей систем бизнес-аналитики заключается в том, что они позволяют выполнять ориентированный на будущее стратегический анализ, например изучать сценарии «что, если». Представьте себе такие вопросы, как: «Если мы предложим продукт на новой платформе, то каких его продаж можно ожидать в будущем?» или: «Если мы будет предлагать клиентам товары в соответствии с их полом, насколько больше они будут покупать?» Система может реализовать модели и алгоритмы, обеспечивающие подобные типы экстраполяции и прогнозирования на основе данных. Эти модели и алгоритмы нужно определять в требованиях к ПО. Если они очень сложные, бизнес-аналитик может вызваться помочь специалистам по данным, математическому моделированию и статистике.

Анализ может требовать статистических и других вычислений для преобразования данных до их представления пользователю или целевой системе для выполнения соответствующих действий. Эти вычисления могут определяться бизнес-правилами в организации или отраслевыми стандартами. Например, если анализ предусматривает получение отчетности о рентабельности валовой прибыли по регионам, нужно точно определить, как эта рентабельность вычисляется в организации. Определенная Стивеном Уитхоллом схема требования для формулы вычисления может применяться практически для любых вычислений, необходимых для преобразования данных (Withall, 2007). Указанная формула должна включать описание вычисляемого значения, саму формулу, используемые переменные и откуда поступают их значения. Также определите все применимые требования по времени отклика этих вычислений.

Эволюционная природа аналитики

Рис. 25-1 иллюстрирует двусторонние связи между данными, их анализом и использованием (Franks, 2012). Время от времени пользователь получает от-

чет и принимает решение. Но чаще в приложениях бизнес-аналитики пользователь начинает с вопроса и запрашивает отчет, содержащий информацию, относящуюся к решению, которое надо принять. Кто-то извлекает нужные данные из доступных хранилищ, применяет соответствующую аналитическую обработку и предоставляет отчет пользователю. Но после получения этой информации у пользователя могут возникнуть новые вопросы, требующие дальнейшего анализа, что ведет к запросу новых отчетов и дополнительного анализа.

Таким образом, при определении требований в проектах аналитики нужно с чего-то начать. Так как требования могут меняться со временем, начните с той информации, которую заинтересованные лица точно хотят получать из существующих данных, и рассчитывайте на то, что их вопросы будут расширяться. Также нужно понять ожидания пользователей в плане расширения своих потребностей. Например, если они ожидают, что их требования к решению бизнес-аналитики будут значительно меняться со временем, им может требоваться решение, которое легко адаптировать с минимальными усилиями по разработке.

Решение аналитики также должно учитывать формы и условия, в которых могут находиться данные в момент извлечения из источника, а также анализироваться и просматриваться пользователем. Например, хотят ли пользователи получать определенные сырые данных, чтобы самостоятельно генерировать отчеты вручную? Или они хотят, чтобы приложение упорядочивало для них данные и предоставляло в заданном структурированном формате? Есть ли у пользователей ряд вопросов, на которые они хотят получать ответы еженедельно на протяжении следующего года? Или они хотят иметь возможность задавать вопросы ежедневно, быстро разрабатывая новые формы анализа и представления данных, чтобы поспевать за быстрыми изменениями в бизнес-потребностях? Ответы на подобные вопросы позволят разработчикам понять, нужно ли предоставлять пользователям данные для самостоятельной работы, или генерировать и форматировать новую информацию для таких пользователей будет команда аналитиков (Franks, 2012).

Ваша задача как бизнес-аналитика в проекте бизнес-аналитики заключается во взаимодействии с заинтересованными лицами проекта над тем, чтобы разобраться с процессами принятия решений. Используйте эти решения для выявления требований, предусматривающих доступ к нужным данным, определяющих, какой анализ надо выполнять и как должны представляться данные. Вы должны понимать, какие результаты заинтересованные лица ожидают от решения аналитики, какие решения, по их мнению, оно должно помогать им принимать и как они хотят динамически менять процедуру анализа и представление его результатов. Изыщите возможности помочь пользователям добиться большего успеха в выработке решений, о которых они даже не могли себе представить.

Глава 26

Проекты встроенных и других систем реального времени

До этого момента большая часть примеров требований и рассуждений относилась к информационным бизнес-системам. Но в мире много продуктов, в которых для управления устройствами используется ПО, к которому применяется общий термин «встроенные системы». Среди бесчисленного множества таких систем сотовые телефоны, телевизионные пульты дистанционного управления, самые разные терминалы, интернет-маршрутизаторы и роботизированные автомобили. Мы часто в этой книге использовали термин *система* как разговорный синоним продукта, приложения или решения для обозначения содержащей ПО вещи, которую планируется создать. Но в этой главе система обозначает продукт, содержащий множественные интегрированные программные и аппаратные подсистемы. Управляющее системой реального времени ПО может внедряться в устройство в форме выделенного компьютера или располагаться на компьютере отдельно от оборудования, которым оно управляет. Встроенные и другие системы реального времени имеют датчики, контроллеры, моторы, подсистемы электропитания, интегральные схемы и другие механические, электрические и электронные компоненты, работающие под управлением ПО.

Системы реального времени делятся на аппаратные и программные. *Аппаратные системы реального времени* отличаются жесткими ограничениями по времени. Выполняемые системой операции должны выполняться в жесткие сроки, иначе возможны серьезные неприятности. Жизненно важные и критически важные для безопасности системы, такие как системы управления воздушным движением, являются аппаратными системами реального времени. Незавершенная вовремя операция может привести к столкновению с не обнаруженным вовремя препятствием. *Программные системы реального времени* подчиняются ограничениям по времени, но последствия нарушения сроков выполнения определенных операций менее катастрофичны. Банкомат — пример программной системы реального времени. Если обмен информацией между банкоматом и банком не завершится в выделенный временной интервал, никто не умрет, если банкомат попытается повторить попытку или завершит операцию.

Больше чем в каких-либо других проектах разработки ПО, в таких проектах важно хорошо понимать требования до того, как разработка встроенных систем зайдет слишком далеко. Так как ПО более гибко, чем оборудование, дополнительные усилия по разработке требований при изменении оборудования обходятся дороже, чем сравнимые изменения в чисто программных проектах. Важно также знать об ограничениях, которые должны соблюдать разработчики как оборудования, так и ПО: размеры физических объектов, электрические компоненты, подключения и вольтаж, стандартные протоколы связи, обязательная последовательность определенных операций и т. п. Уже выбранные для проектирования аппаратные компоненты накладывают ограничения на еще не выбранные узлы.

Ясно, что описанные в других частях этой книги приемы выявления требований вполне применимы к проектам реального времени. Можно использовать те же приемы моделирования с небольшими корректировками. В этой главе рассказывается об особенностях требований к встроенным и другим системам реального времени.

Системные требования, архитектура и назначение

При определении сложных систем многие команды начинают со спецификации системных требований, или SyRS (ISO/IEC/IEEE, 2011). SyRS описывают возможности системы в целом, в том числе те, что могут обеспечиваться аппаратными компонентами, программными компонентами и/или людьми. Они также описывают весь связанный с системой ввод и вывод. Помимо функциональности SyRS должны определять требования по критической производительности, безопасности и другим требованиям к качеству продукта. Вся эта информация ложится в основу предварительного анализа дизайна и будет служить команде ориентиром при выборе архитектурных компонентов и назначении им функций. SyRS могут поставляться отдельно от спецификации требований к ПО или спецификация может внедряться в SyRS, особенно если основная сложность системы заключена в ПО.

Анализ требований сложных систем тесно связан с их архитектурой. В системах реального времени обдумывание требований и дизайна связаны теснее, чем в других проектах разработки ПО. Архитектура представляет верхний уровень дизайна, изображаемый в виде блок-схемы с прямоугольниками и стрелками, хотя существует много других нотаций моделирования архитектуры. Архитектура системы состоит из трех элементов:

- Компонентов системы, под которыми могут подразумеваться программные объекты или модули, физическое устройство или человек;
- Видимые снаружи свойства компонентов;
- Подключения (интерфейсы) между компонентами системы.

Архитектура разрабатывается итеративно в направлении сверху вниз (Nelsen, 1990; Hooks и Farry, 2001). Человек, играющий ведущую роль в анализе такого типа, обычно является бизнес-аналитиком, инженером требований, системным инженером или системным архитектором с обширными техническими знаниями. Аналитик разбивает систему на программные и аппаратные подсистемы и компоненты, которые будут обеспечивать весь вход и выход. Некоторые системные требования напрямую преобразуются в требования к ПО, если ПО оказывается тем агентом, который обеспечивает нужную функцию. В других случаях аналитик должен разбить отдельные системные требования на много производных требований к ПО, оборудованию или требований к ручному управлению, выполняемому человеком (рис. 26-1). Выведение требований к ПО из системных требований может многократно увеличить число требований, отчасти еще из-за того, что при этом создаются требования к интерфейсам между компонентами. Аналитик назначает конкретные требования наиболее подходящим компонентам, итеративно уточняя разбиение архитектуры и назначение требований. В конечном итоге получается набор требований для всех программных и аппаратных компонентов, а также людей, участвующих в предоставлении нужных системных сервисов.

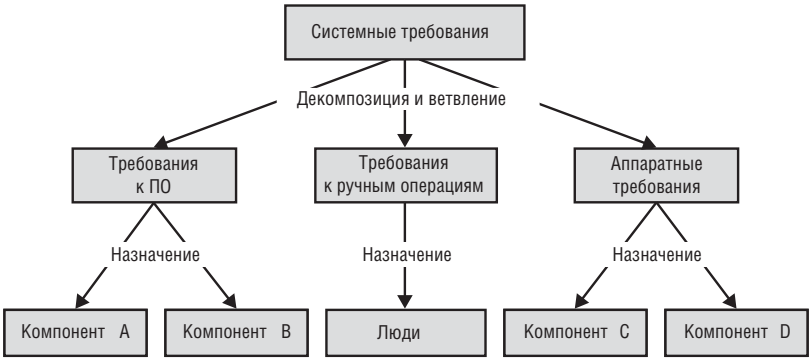


Рис. 26-1. Системные требования разбиваются на ряд требований для всех программных и аппаратных компонентов, а также для людей, которые далее назначаются соответствующим компонентам

Желательно устанавливать связи между системными требованиями, производными требованиями к ПО и оборудованию, а также архитектурными компонентами, на которые они назначены. Подробнее об отслеживании связей требований см. главу 29.

Неудачные решения по назначению требований могут приводить к следующим результатам:

- Выполнение функций возлагается на ПО, хотя дешевле и проще реализовать их с помощью соответствующего оборудования (или наоборот);
- Выполнение функций возлагается на человека, хотя дешевле и проще реализовать их с помощью соответствующего оборудования (или наоборот);

- Неудовлетворительная производительность;
- Невозможность просто обновлять или заменять компоненты.

Например, выполнение определенной функции средствами ПО может потребовать более быстрого процессора, тогда как специализированное аппаратное обеспечение может выполнить эту функцию с существующим процессором. Всегда возникают компромиссы. Хотя ПО само по себе более гибко, чем оборудование, инженерам не следует использовать это качество, чтобы сэкономить на дизайне оборудования. Специалисты, назначающие требования, должны понимать возможности и ограничения программных и аппаратных компонентов, а также стоимость и риски реализации функциональности в каждом из них.

Моделирование систем реального времени

Как и в случае информационных бизнес-систем, визуальное моделирование является мощным приемом анализа при определении систем реального времени. Для этого больше всего подходят диаграммы переходов состояний или их более сложные варианты, такие как блок-схемы состояний (Lavi и Kudish, 2005) и диаграммы машины состояний UML (Ambler, 2005). Брюс Пауэлл Дуглас (Douglass, 2001) дает примеры, как использовать варианты использования и другие UML-модели для представления требований к системам реального времени.

Большинство систем реального времени существуют во многих состояниях с определенными условиями и событиями, разрешающими переходы между состояниями. Таблицы состояний и решений могут дополнять или заменять диаграммы переходов состояний, подчас позволяя обнаружить ошибки в диаграммах. Контекстные диаграммы (описаны в главе 5) также полезны для представления среды, в которой работает система, и границ между системой и внешними сущностями, с которыми она взаимодействует. Архитектурные диаграммы показывают разбиение системы на подсистемы с интерфейсами между ними. В следующем разделе приводятся несколько примеров моделей (как обычно, немного упрощенных) встроенной системы, с которой у вас возможно был опыт, — беговой дорожки.

Контекстная диаграмма

На рис. 26-2 показана контекстная диаграмма моей домашней беговой дорожки. Нотация здесь немного отличается от показанной ранее в главе 5 на рис. 5-6, но цель и типы информации те же (Lavi и Kudish, 2005). Использование большого прямоугольника вместо небольшого круга для представления системы облегчает представление множественных входных и выходных потоков между системой и отдельными внешними сущностями, например с Бегуном (человеком на беговой дорожке). Другими двумя внешними сущностями являются веб-сайт производителя беговой дорожки, откуда Бегун может загрузить различные программы тренировок, и датчик измерения частоты пульса Бегуна. Как обычно на контекстных диаграммах, здесь нет ничего о внутренней структуре беговой дорожки.

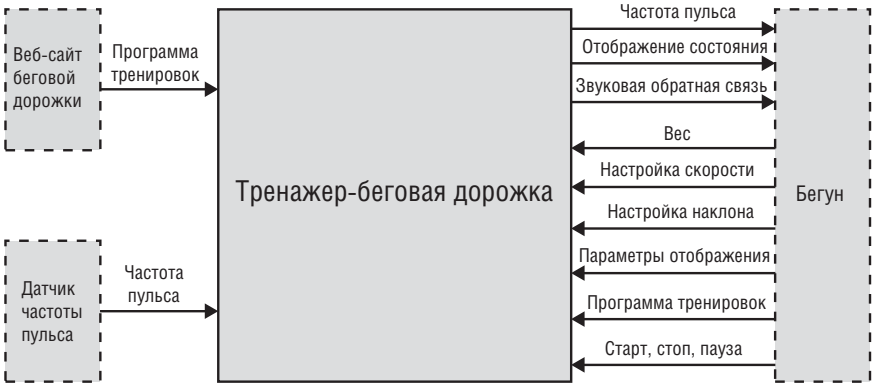


Рис. 26-2. Контекстная диаграмма тренажера-беговой дорожки

Диаграмма переходов состояний

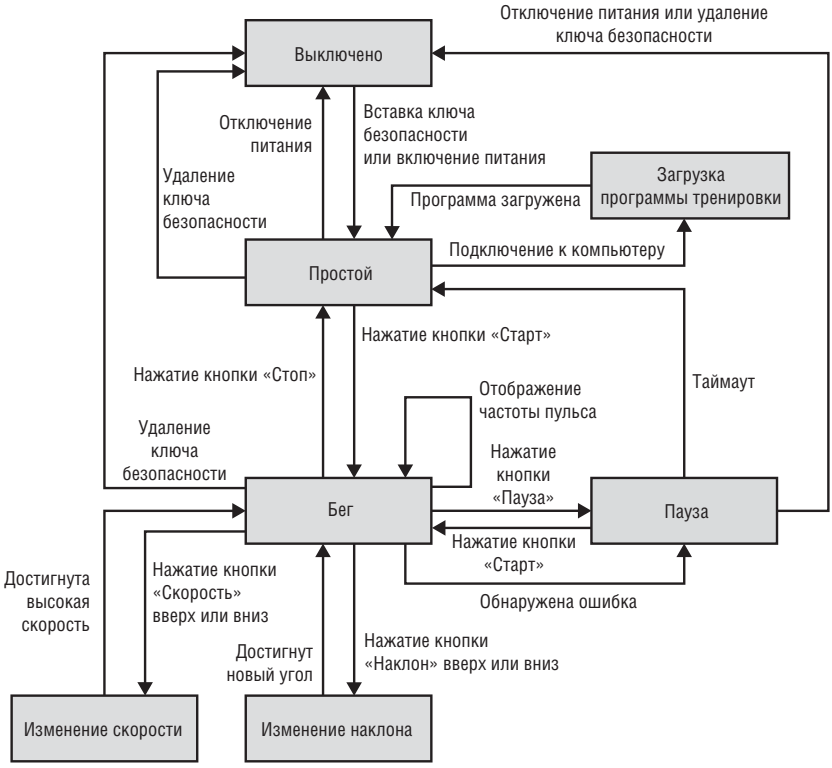


Рис. 26-3. Частичная диаграмма переходов состояний тренажера-беговой дорожки

На рис. 26-3 показана диаграмма переходов состояний беговой дорожки. Как вы помните из главы 12, блоки на диаграмме переходов состояний представляют различные состояния, в которых может оказаться беговая дорожка, а стрелки представляют разрешенные переходы между ними. Надписи

на стрелках перехода указывают на состояния или события, инициирующие каждое изменение состояния. Эта диаграмма дает больше информации о работе тренажера. Она также начинает предоставлять определенную информацию о необходимых элементах управления на пользовательском интерфейсе, в частности «Скорость», «Наклон», «Старт», «Пауза» и «Стоп». На рис. 26-3 говорится о «нажатии» элементов управления, но они могут быть реализованы не в виде кнопок. Йохан Лави и Джозеф Кудиш (Lavi и Kudish, 2005) описывают более сложные диаграммы состояний, для представления этой информации в более подробном виде.

Таблица событий и реакций

Анализ событий и реакций является еще одним способом проанализировать поведение системы реального времени и ее функциональных требований (Wiley, 2000). Как говорилось в главе 12, система может реагировать на бизнес-события, которые инициируют выполнение варианта использования, сигнализируют о событиях, таких как входные данные с датчика или временны*е* события, которые приводят к выполнению какой-то операции после заданного интервала времени или в определенный момент времени. В табл. 26-1 перечислены несколько событий и реакция на них тренажера.

Табл. 26-1. Фрагмент таблицы событий и реакций для тренажера-беговой дорожки

Событие	Состояние тренажера	Реакция
Бегун нажимает вверх кнопку «Наклон»	Ниже максимального наклона	Увеличение наклона на полградуса
Бегун нажимает вверх кнопку «Наклон»	Наклон максимальный	Генерация звукового сигнала «крайнее положение»
Бегун нажимает кнопку «Скорость» вниз	Скорость выше минимальной	Снижение скорости на 0,1 м/час
Бегун нажимает кнопку «Скорость» вниз	Скорость минимальная	Остановка беговой ленты
Бегун удаляет ключ безопасности	Бег	Остановка беговой ленты и отключение питания
Бегун удаляет ключ безопасности	Простой	Отключение питания
Бегун нажимает кнопку «Пауза»	Бег	Остановка беговой ленты, включение таймера
Бегун нажимает кнопку «Пауза»	Пауза или Простой	Генерация звукового сигнала «ошибка»
Таймер состояния паузы достигает времени таймаута	Пауза	Переход в состояние «Простой»

Табл. 26-1. (*окончание*)

Событие	Состояние тренажера	Реакция
Бегун нажимает кнопку «Старт»	Бег	Генерация звукового сигнала «ошибка»
Бегун нажимает кнопку «Старт»	Пауза	Включить беговую ленту с текущей скоростью
Бегун нажимает кнопку «Старт»	Простой	Включить беговую ленту на самой низкой скорости

Этот список событий предоставляет подробные требования к функциональности беговой дорожки, который лежат в основе высокоуровневого представления на рис. 26-3. Это также отличный материал для создания тестов. Даже полная таблица событий и реакций все равно оставляет много простора для размышлений при дизайне, например на сколько градусов в минуту должен менять наклон дорожки мотор изменения наклона и как быстро дорожка должна разгоняться с нулевой до заданной скорости. Эти решения будут также определяться соображениями безопасности. Слишком резкий старт, ускорение или остановка могут быть небезопасными для Бегуна.

Встроенные системы должны управлять сочетанием основанных на событиях функций (как показано в табл. 26-1) и функций периодической проверки. Периодические функции выполняются циклически, когда система находится в определенном состоянии, а не однократно при переходе в определенное состояние. В качестве примера можно привести измерение частоты пульса Бегуна раз в секунду и корректировка скорость дорожки, чтобы обеспечить заданную частоту пульса — если, конечно, такую программу кто-то решит реализовать.

Рисование таких моделей — отличный способ обнаружения пропущенных требований. Как-то мне пришлось рецензировать спецификацию требований к встроенной системе, которая содержала длинную таблицу, описывающую различные состояния машины, относящуюся к каждому состоянию функциональность и возможные направления перехода из каждого состояния. Я нарисовал диаграмму переходов состояний, чтобы представить эту информацию на более высоком уровне абстракции. В процессе построения диаграммы я обнаружил два отсутствующих требования. Не было требования, предусматривающего отключение машины, а также не была предусмотрена возможность перехода в состояние ошибки из рабочего состояния машины. Как вы видели раньше, этот пример иллюстрирует ценность создания множественных представлений знания о требованиях и перекрестной их перепроверки.

Архитектурная диаграмма

Другой тип модели, который полезен для систем такого типа, — архитектурная диаграмма, которая обычно является частью высокоуровневого дизайна. На рис. 26-4 показан фрагмент простой архитектурной диаграммы беговой

дорожки. Она определяет основные подсистемы, обеспечивающие все функции тренажера, а также данные и интерфейсы управления между ними на высоком уровне абстракции. Существуют более богатые языки описания, а также для моделирования архитектур можно использовать UML (Unified Modeling Language) (Rozanski и Woods, 2005). Показанные на рис. 26-4 подсистемы в процессе анализа архитектуры могут в дальнейшем становиться конкретными аппаратными (моторы и датчики) и программными компонентами. Предварительный анализ архитектуры может обнаружить и уточнить функциональные требования, а также требования к интерфейсам и качеству, которые не удалось обнаружить в процессе других мероприятий по выявлению требований.

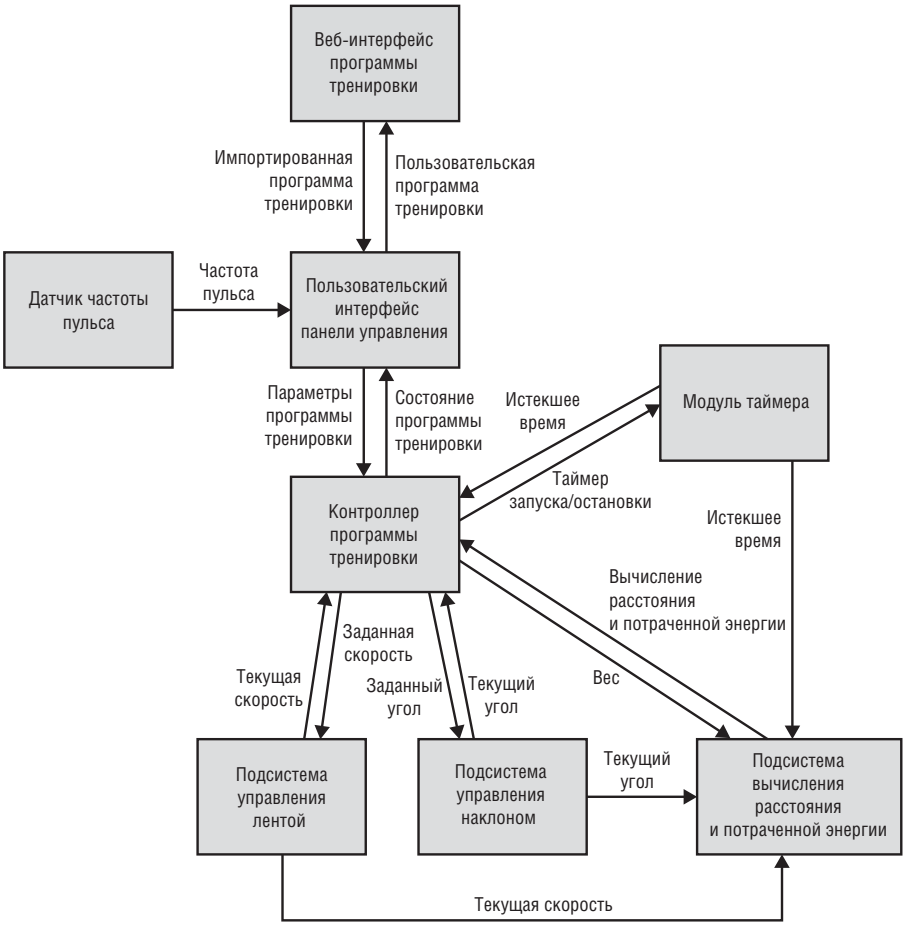


Рис. 26-4. Фрагмент архитектурной диаграммы тренажера

Рисование архитектурных моделей в процессе анализа требований — четкая переходная фаза к дизайну. Это необходимый этап. Последовательно проходя архитектурное разбиение и назначение функций подсистемам и компонентам, архитектор создает наиболее подходящее и эффективное решение.

Но требуется дополнительное выявление требований. Функциональные требования, наподобие приведенных ниже, позволяют разработчикам выбрать правильные компоненты оборудования и спроектировать элементы пользовательского интерфейса:

Incline.Angle.Range *У бегуна должна быть возможность увеличивать и уменьшать угол наклона беговой дорожки от нуля до 10 градусов включительно с шагом в полградуса.*

Incline.Angle.Limits *Беговая дорожка должна прекратить изменение угла и дать звуковой сигнал при достижении минимального или максимального значения наклона.*

Помимо представленной в этой архитектуре функциональности проектировщики беговой дорожки должны знать о бизнес-правилах, обеспечивающих необходимые алгоритмы. В качестве примера можно привести вычисление количества калорий, потраченных Бегуном, на основе его веса и выбранной программы тренировок, которая состоит из серии продолжительности периодов времени, углов наклона и скорости беговой ленты. Разговор о бизнес-правилах в применении к встроенной системе может показаться странным. Однако практически все приемы работы с требованиями, описанные в других главах этой книги, применимы к встроенными и другими системам реального времени — точно так же, как к информационным бизнес-системам.

Создание прототипов

Создание прототипов и имитация — мощные приемы выявления и проверки требований к встроенным системам. Так как создание оборудования (или его повторное построение в случае выявления ошибок) требует значительных затрат денег и времени, для тестирования оперативной концепции можно задействовать прототипы и исследовать требования и варианты дизайна устройства. Имитация может помочь лучше понять внешний вид и элементы управления пользовательского интерфейса, сетевые взаимодействия и интерфейсы между оборудованием и ПО (Engblom, 2007). Но надо не забывать, что имитация во многом отличается от реального продукта.

Интерфейсы

Интерфейс — критически важная часть встроенных и других систем реального времени. Как мы видели в главе 10, спецификация требований должна охватывать четыре класса требований к внешним интерфейсам: пользователи, ПО, оборудование и коммуникационные интерфейсы. Кроме того секционирование сложны систем на ряд подсистем создает множественные внутренние интерфейсы между компонентами. В рамках большого продукта встроенные системы могут встраиваться в другие встроенные системы (например, мобильный телефон интегрируется в коммуникационную систему

автомобиля), что дополнительно усложняет вопросы интерфейса. Анализ требований должен сосредоточиться на вопросах внешних интерфейсов, оставив спецификации внутренних интерфейсов для проектирования архитектуры.

Если внешние интерфейсы относительно простые, можно определить их как описано в разделе 5 спецификации требований к ПО на рис. 10-2 в главе 10. В проектах создания сложных систем часто создается отдельная спецификация для документирования этих критически важных вещей. На рис. 26-5 предлагается шаблон для документа спецификации интерфейсов, который может содержать информацию как внешних, так и внутренних интерфейсов.

1. Введение
   - 1.1 Цель документа
   - 1.2 Обзор продукта
   - 1.3 Операционная среда
   - 1.4 Ссылки
   - 1.5 Предположения
2. Диаграммы интерфейсов
3. Интерфейсы данных
   - 3.x <Идентификатор интерфейса x>
     - 3.x.1 Обзор
     - 3.x.2 Типы данных
     - 3.x.3 Форматы файлов интерфейсов
     - 3.x.4 Коммуникационный протокол
4. Программные интерфейсы
   - 4.x <Идентификатор интерфейса x>
     - 4.x.1 Обзор
     - 4.x.2 Спецификация интерфейсов
     - 4.x.3 Вопросы управления временем
     - 4.x.4 Коммуникационный протокол
5. Аппаратные интерфейсы
   - 5.x <Идентификатор интерфейса x>
     - 5.x.1 Обзор
     - 5.x.2 Подключение
     - 5.x.3 Поток данных и управления
6. Пользовательские интерфейсы

Рис. 26-5. Предлагаемый шаблон спецификации интерфейсов

Требования к временным характеристикам

Требования к временны*м* параметрам — краеугольный камень систему управления реального времени (Koopman, 2010). Возможны нежелательные последствия, если сигналы поступают от датчиков не когда запланировано, если программа не может отправить сигналы управления на устройство, когда это нужно, или если физические устройства не выполняют свои действия вовремя. У требований к временным характеристикам несколько измерений:

- **Время выполнения** Это время, которое проходит от запуска задачи до ее завершения. Оно может измеряться как время между двумя определенными событиями, увязывающими выполнение задачи.

- **Задержка** Это время между моментом запуска события и реакции на него системы. В частности, слишком большие задержки являются проблемой в записи музыки или производстве ПО, в котором требуется точная синхронизация множества готовых и живых аудиозаписей.
- **Предсказуемость** Она подразумевает повторяемое и последовательное время события. Даже если выполнение по времени не слишком «быстрое», часто действия должны выполняться в точно определенные моменты, например при измерении входного сигнала. Оцифровка аудиосигнала часто выполняется с частотой 44 100 цикла в секунду. Частота дискретизации должна быть прогнозируемой, чтобы не было искажений оцифрованного аудиосигнала.

Вот вопросы, которые нужно исследовать при определении требований к временным характеристикам и к графикам задач систем реального времени:

- Периодичность (частота) выполнения задач и допуски.
- Сроки и допуски выполнения каждой задачи.
- Типичное и худшее время выполнения каждой задачи.
- Последствия пропуска срока.
- Минимальное, среднее и максимальное время прихода данных в каждом соответствующем состоянии компонента.
- Максимальное время ожидания первого ввода или вывода после запуска задачи.
- Что делать, если через максимальное время ожидания данные не получены к моменту первого ввода (таймаут).
- Обязательная последовательность выполнения задач.
- Задачи, выполнение которых должно начаться или закончится до запуска других задач.
- Приоритеты задач, чтобы знать, какие задачи могут прерывать или вытеснять другие и на каких основаниях.
- Функции, зависящие от текущего состояния системы (например, нормальный режим и режим пожарной тревоги у лифта).

При определении требований по времени, укажите все ограничения и приемлемые допуски времени. Разберитесь с различием между программными и аппаратными запросами реального времени в вашей системе, чтобы не задать слишком жестких требований по времени. Это может привести к чрезмерному техническому усложнению продукта и сопутствующим дополнительным затратам. При более мягких допусках по времени можно обойтись менее дорогим оборудованием. Как говорит Филип Коопман (Koopman, 2010): «Производительность реального времени редко означает, что нужна максимально возможная скорость — дело в скорости, которая необходима, и минимизации общих затрат».

Для определения требований по времени для системы нужно понимать сроки критически важных по времени функций. Это подразумевает плани-

рование как последовательных, так и одновременных функций для достижения нужной производительности в рамках ограничений возможностей процессора, скоростей ввода/вывода и быстродействия сети. Одна команда применяла средство для составления графика моделирования временных требований для встроенного ПО, причем за единицу измерения были приняты миллисекунды, а не более привычные дни или недели. Такое изобретательное и нетрадиционное использование средства моделирования сработало отлично. В некоторых случаях алгоритмы назначения времени и графика могут налагаться через требования в форме ограничений дизайна, но чаще это бывает выбор определенного дизайна. Кришна Кави, Роберт Акл и Али Хурсон предлагают ценный обзор вопросов планирования в системах реального времени (Kavi, Akl и Hurson, 2009).

Атрибуты качества для встроенных систем

Требования к атрибутам качества особенно важны для встроенных и других систем реального времени. Они могут быть намного более сложными и взаимосвязанными, чем другие программные приложения. Бизнес-программы обычно используются в офисе, где среда довольно однообразна. С другой стороны, рабочая среда встроенных систем может подразумевать крайние температуры, вибрацию, удары и другие условия, которые диктуют определенные соображения качества. Категории качества, которые скорее всего будут особенно важными, включают производительность, эффективность, надежность, устойчивость, безопасность, защита и удобство использования. В этом разделе обсуждаются некоторые аспекты этих атрибутов качества, которые нужно внимательно изучить в процессе выявления требований к таким системам.

Помимо атрибутов качества ПО, которые обсуждались в главе 14, во встроенных системах надо учитывать атрибуты качества и ограничения, которые применяются к физическим системам. Среди них размер, форма, вес, материалы, огнеопасность, подключения, долговечность, стоимость, уровень шума и прочность. Все это может сильно увеличить затраты и усилия необходимые на адекватную проверку требований. Могут быть политические или обусловленные бизнесом причины избегать материалов, поставки которых могут быть нарушены конфликтом или бойкотом, при которых подскакивают цены. Есть материалы, которых следует избегать из-за их неэкологичности. Неиспользование оптимальных материалов может приводить к компромиссам в производительности, весе, затратах и других атрибутах.

Может быть сложно или дорого реализовать желаемую характеристику качества после завершения дизайна оборудования, поэтому нужно определить эти требования на ранних этапах выявления. Так как характеристики качества часто оказывают глубокое влияние на архитектуру сложного продукта, важно определять приоритеты атрибутов и анализ компромиссов до начала дизайна. Коопман (Koopman, 2010) приводит отличное обсуждение нефункциональных требований, которые особенно важны для разработки

встроенных систем. В главе 14 приведено много примеров этих и других требований к атрибутам качества.

Производительность Суть системы реального времени заключается в том, что ее производительность должна удовлетворять требованиям по времени и ограничениям рабочей среды. Поэтому все сроки обработки конкретных операций должны быть включены в требования. Однако производительность означает больше, чем время реакции в процессе работы, и включает время загрузки и перезагрузки, энергопотребление, время работы от батареи, время зарядки батареи (например, в электромобилях) и рассеяние тепла. У одного лишь управления энергоснабжением масса нюансов. Как должна вести себя система в случае моментального падения напряжения, или при особенно высоком токе при загрузке, или при пропадании электропитания и необходимости перейти на запасной источник энергии? И, в отличие от ПО, рабочие характеристики многих компонентов системы могут ухудшаться. Каковы требования к времени, на протяжении которого батарея поддерживает определенный план электропитания до замены?

Эффективность Это внутренний двойник наблюдаемого снаружи атрибута производительности. Эффективность встроенных систем связана с потреблением (а, значит, с остатком на данный момент) ресурсов, в числе которых мощность процессора, память, дисковое пространство, каналы связи, электропитание и пропускная способность сети. При работе с этими вещами становится заметной тесная связь между требованиями, архитектурой и дизайном. Например, если общая потребность в электропитании устройства превышает имеющиеся мощности, можно ли изменить дизайн так, чтобы питание не используемых постоянно компонентов отключалось, освобождая ресурсы для других компонентов или сервисов?

Требования должны указывать максимальное ожидаемое потребление различных системных ресурсов, чтобы дизайнеры могли предусмотреть нужный запас ресурсов в расчете на рост в будущем и неожиданные условия работы. Это одна из тех ситуаций, когда параллельное проектирование оборудования и ПО жизненно важны. Если ПО потребляет слишком много имеющихся ресурсов, разработчикам приходится прибегать к сложным уловкам, чтобы обойти эти ограничения. Выбор более мощного оборудования позволяет значительно удешевить продукт, не прибегая к дорогой доработке ПО (Koopman, 2010).

Надежность Встроенные и другие системы реального времени часто отличаются жесткими требованиями к надежности и доступности. Жизненно важные системы, такие как медицинские устройства и авионика самолетов не оставляет места для отказа. Вшитый в тело пациента кардиостимулятор должен надежно работать на протяжении многих лет. Если он откажет или батарейка истощится раньше времени, пациент может умереть. При определении требований к надежности нужно реалистично оценивать вероятность и влияние отказа, чтобы чрезмерно не усложнять продукт, требования к надежности которого могут оказаться не такими высокими, как кажется.

Повышение надежности и доступности не бесплатное удовольствие. Иногда приходится платить эту цену, но есть случаи, когда это неразумно.

День открытых дверей

В одном из больших городов Америки при отправлении поезда монорельсовой железной дороги от станции не закрылась дверь вагона. Видимо датчики не уведомили водителя поезда о неисправности. Поезд мчался со скоростью 90 км/час с открытой дверью — довольно пугающее зрелище и очевидная опасность для жизни. У разработчиков ПО для поезда скорее всего было требование к надежности и защите, где говорилось, что такое событие может произойти не чаще одного раза на каждые 100 млн часов работы. Невозможно перед выпуском прогнать поезд несколько сотен миллионов часов, чтобы проверить выполнение требования. Вместо этого системы надо проектировать так, чтобы вероятность критически опасного отказа была достаточно низкая, чтобы удовлетворить требование. Но вещи ломаются. В таких сложных системах причиной обычно становится сочетание отказов, о котором не подумали, — в данном случае причиной такого редкого отказа стала коррозия сразу двух переключателей.

Устойчивость Это характеристика того, как система справляется с неожиданными условиями работы. У надежности есть несколько аспектов. Один из них — живучесть, которая часто применяется в анализе устройств не только для военного применения, но и для предметов повседневного применения. Хороший пример встроенной системы, сконструированной в расчете на высокую живучесть, — «черный ящик» самолета, который должен сохраниться в самых серьезных авариях. На самом деле ярко-оранжевый ящик, который официально называется аварийный бортовой самописец и должен выдерживать удары в 3400g, пожар, погружение в воду и другие опасности. В таких условиях должна сохраняться целостность не только физического контейнера, но устройства записи и сами записи должны оставаться в целостности и быть читабельными.

Еще одна сторона устойчивости характеризует то, как система справляется с ошибками или исключениями, которые происходят во время работы и могут вести к отказу системы. Отказы системы могут вызываться сбоями как оборудования, так и ПО. Однажды я попытался получить в банкомате 140 долларов. Банкомат без проблем распечатал квитанцию на 140 долларов, но выдал только 80. Мне пришлось ждать около 15 минут, пока сотрудник банка ковырялся в банкомате, после чего выдал мне недостающие 60 долларов. Очевидно, сбой имел механическую природу: несколько банкнот склеились и заблокировали щель выдачи. Я не только потратил время, но и поволновался — ведь банкомат «полагал», что все в порядке и никаких проблем не обнаружил.

Есть четыре характеристики того, как система обрабатывает сбои (Koopman, 2010):

- **Предотвращение сбоев** В идеале система должна предотвращать сбои, прежде чем они смогут привести к отказу. В основе этого принцип проверки программой предварительных условий перед инициированием выполнения варианта использования.
- **Обнаружение сбоев** Следующая мера — обнаружение сбоя, как только он возникает. Вот почему при выявлении требований нужно изучать исключения, чтобы разработчики могли учитывать возможные ошибки и предусматривать проверку их наличия.
- **Восстановление после сбоя** В системе должны быть механизмы реагирования на случай обнаружения ожидаемой ошибки. При разработке требований нужно не только определить возможные сбои, но и то, как на них реагировать. Иногда система может попытаться повторно выполнить операцию, как, к примеру, в случае ненадежного соединения, или после определенного времени таймаута. В системах иногда предусмотрен механизм перехода при ошибке. Если после сбоя система терпит отказ, выполнение работы берет на себя запасная система. В других случаях система должна прекратить работу, например выключиться или перезагрузиться, чтобы снизить негативное влияние на пользователя. Например, если антиблокировочная система тормозов автомобиля (АБС) обнаруживает неисправный датчик, она может выключить АБС и включить предупредительный световой сигнал на инструментальной панели, а также зарегистрировать это событие в компьютере автомобиля для дальнейшей диагностики и ремонта. Это подводит нас к следующей характеристике.
- **Регистрация сбоев** В системе должна храниться история обнаруженных сбоев и их последствий. Эта информация полезна для диагностики неполадок, и позволяет обслуживающему персоналу обнаруживать шаблоны поведения, приводящие к проблемам. Например, история сбоев может указывать на дефектный аппаратный компонент, который надо заменить. У современных автомобилей есть бортовая диагностическая система. Мастер может подключить к системе кабель и получить историю событий в форме стандартизованных кодов ошибок в системе.

Проектировщики моей беговой дорожки знали, что при определенных условиях беговая дорожка может заблокироваться в положении, в котором угол наклона нельзя уменьшить до нуля. В руководстве пользователя описывается (довольно непростая) ручная операция, которую я могу выполнить для сброса настройки беговой дорожки, чтобы снова стал доступен весь диапазон углов наклона. Конечно, было бы лучше, если бы производитель сконструировал тренажер так, чтобы его нельзя было заблокировать в определенном положении. Но иногда предоставить обходное решение для маловероятных и не очень серьезных сбоев оказывается намного дешевле, чем полностью предотвратить их.

Защита Любая система, где есть движущиеся части или где используется электричество, может причинить увечье или даже привести к смерти человека. Требования к защите намного более важны для систем реального вре-

мени, чем для информационных систем. Проектированию безопасного ПО и систем посвящено много книг, поэтому мы не будем здесь повторять всю эту жизненно важную информацию. Мы отошлем вас к таким, хорошим авторам, как Ненси Левенсон (Leveson, 1995), Дебра Херрманн (Herrmann, 1999), Филип Коопман (Koopman, 2010) и Терри Харди (Hardy, 2011).

Начните свое изучение требований к защите, выполнив *анализ опасностей* (Ericson, 2005; Ericson, 2012). Это позволит выявить возможные опасности, которые может представлять ваш продукт. Их можно упорядочить по вероятности возникновения и серьезности последствий, что позволит сосредоточиться на самых серьезных опасностях. (Подробнее об анализе рисков см. главу 32.) *Анализ дерева неисправностей* (fault tree analysis) представляет собой графический прием анализа в стиле «причина-следствие» для определения опасностей нарушения защиты и факторов, ведущих к ним (Ericson, 2011). Это позволяет сосредоточиться на том, как избежать материализации определенных сочетаний факторов риска при использовании вашего продукта. Требования к защите должны предусматривать смягчения рисков и указывать, что система должна — или не должна делать, чтобы избежать этих рисков.

У физических устройств обычно есть что-то типа аварийной кнопки, которая быстро их выключает. У беговой дорожки есть требование к безопасности, которое звучит примерно так:

Stop.Emergency *У беговой дорожки должна быть механизм аварийного останова, который останавливает ленту беговой дорожки в течение одной секунды после приведения его в действие.*

Это требование привело к созданию плоского пластикового ключа, который надо вставить в щель в передней части беговой дорожки, чтобы ее можно было включить. При удалении ключа питание тренажера отключается, что приводит к быстрой остановке ленты беговой дорожки. Привязанный к ключу шнур можно прикрепить к одежде Бегуна, чтобы ключ вытаскивался при падении или соскальзывании бегуна с беговой дорожки. И это работает!

Безопасность Безопасность встроенных систем сейчас активно обсуждается из-за опасностей, связанных с кибератаками, в результате которых может перехватываться управление, нарушаться работа или отключаться электростанции, системы управления железнодорожным движением, электросети и другие критически важные инфраструктуры. Кража интеллектуальной собственность из памяти встроенных систем также представляет серьезный риск. Атакующий в принципе может выполнить обратную инженерию кода, чтобы узнать, как работает система, что позволит ему скопировать или атаковать ее. Защита встроенных систем подразумевает часть тех же мер безопасности, в которых нуждаются располагающиеся на сервере информационные системы. Это включает следующее (Koopman, 2010):

- Секретность, в основном через шифрование;
- Аутентификацию, чтобы гарантировать, чтобы доступ к системе был только у уполномоченных пользователей. Обычно это реализуется по-

средством паролей (со всеми недостатками, присущими работе человека с паролями);

- Проверки целостности данных, чтобы вовремя обнаружить попытки вмешательства в систему;
- Приватность данных, например защита от неправомочного отслеживания пользователей через их мобильные устройства с поддержкой GPS.

Но кроме этого встроенные системы могут подвергаться другим типам специфических атак. Это включает попытки злоумышленников получить контроль над системой, перехват электронных коммуникаций, особенно беспроводной связи, а также установка вредных обновлений ПО — обычно это выполняется с применением методов социальной инженерии к легковерным пользователям (многие из нас иногда становятся жертвой таких приемов). Весь перечень вопросов безопасности во встроенных системах огромен и они очень важны (Anderson, 2008). Коопман (Koopman, 2010) и Дейвид и Майк Клайдермахер (Kleidermacher, 2012) дают много советов по обеспечению безопасности встроенных продуктов.

Удобство использования Во многих встроенных системах есть какое-то подобие интерфейса взаимодействия человека и компьютера. К нему применяются общие принципы удобства использования ПО, но могут быть важны другие аспекты удобства, если, к примеру, человек использует физическое устройство в полевых условиях, а не за клавиатурой в офисе. Недавно я перешел от использования мыши, предназначенной для правшей, на симметричную модель. Я все время нечаянно щелкаю правую кнопку мыши безымянным пальцем правой руки. Это тратит мое время и может вести к нежелательной реакции системы.

Экран продуктов, которые предполагается использовать на улице, должен адаптироваться к разному освещению. Однажды у меня был счет в банке, в котором банкомат для обслуживания клиентов из автомобиля был расположен так, что при определенном угле зрения его жидкокристаллический экран было совершенно не видно из-за солнечных бликов. Еще один пример: надев свои поляризованные солнечные очки, я не вижу цифр на своих цифровых часах, если только не поверну запястье под правильным углом, — дело в том, что жидкокристаллические экраны поляризованы.

Некоторые ограничения на удобство использования накладывает закон, например Закон об инвалидах в США, который требует, чтобы определенные системы предоставляли специальные возможности для людей с ограниченными возможностями. Встроенные системы должны быть рассчитаны на пользователей:

- С различной степенью остроты и частотной избирательности слуха (это надо учитывать при проектировании звуковой обратной связи и подсказок);
- С различной степенью остроты и цветовой избирательности (дальтонизм) зрения (это надо учитывать при выборе цвета и размера текста на экране);

- С праворукостью или леворукостью и нарушениями моторики (влияет на способность пользователя правильно нажимать маленькие кнопки или пользоваться сенсорным экраном);
- С разным ростом (учитывайте это при размещении элементов управления, экранов и оборудования);
- С различными родными языками (важно для устройств с распознаванием речи).

Проблемы встроенных систем

Встроенные и другие системы реального времени ставят уникальные задачи, которые отсутствуют в чисто программных системах. Основные принципы и приемы выявления, анализа, спецификации и проверки применимы к обоим типам продуктов. При разработке встроенных систем надо применять инженерный подход, чтобы разработчики ПО и оборудования не оптимизировали одно за счет другого, а также во избежание неприятных проблем с интеграцией. Выбор архитектуры и дизайна более тесно связан с анализом требований, чем в чисто программных системах, отчасти потому что изменять аппаратную часть после ее проектирования и производства намного дороже. Во встроенных системах другие приоритеты ограничений и атрибутов качества, чем в чисто программных системах, и зачастую они более тесно связаны с операционной системой. Тщательная спецификация системных требований, требований к ПО, оборудованию и интерфейсу очень важна для успеха проектов разработки встроенных и других система реального времени.

Часть IV

Управление требованиями

Глава 27

Приемы управления требованиями к ПО

«Я наконец доделала функцию запроса по каталогу различных поставщиков, — сообщила Шери на еженедельной встрече, посвященной состоянию проекта Chemical Tracking System. — Вот это была работа!»

«О, клиент отказался от этой функции две недели назад, — подал реплику менеджер проекта Дейв. — Ты разве не получила новую версию спецификации?»

Шери была озадачена. «Как это понимать, отказался? Эти требования черным по белому написаны вверху страницы в самой последней версии моей спецификации».

«Хм, их нет в моей копии. У меня версия 1.5 спецификации. А у тебя?»

«У меня тоже 1.5, — с явным неудовольствием сказала Шери. — Эти документы должны быть идентичны, но очевидно, это не так. И что, эта функция все еще включена в базовую версию или я потратила впустую 30 часов своей жизни?»

Если вы когда-либо слышали подобный разговор, вы знаете, как расстраивает, если люди тратят впустую время, работая над устаревшими или непоследовательными требованиями. Наличие отличных требований всего лишь часть решения — они также должны хорошо управляться и эффективно доводиться до участников проекта. Управление версиями отдельных требований и их совокупностей — один из базовых аспектов управления требованиями.

В главе 1 говорилось, что создание технических условий можно разделить на разработку требований и управление требованиями. (Некоторые называют всю эту область «управлением требованиями», но мы предпочитаем более узкую трактовку этого термина.) В этой главе рассматриваются основные принципы и приемы управления требованиями. В других главах части IV некоторые приемы управления требованиями описываются более подробно, в том числе управление изменениями (глава 28), анализ влияния изменений (глава 28) и отслеживание требований (глава 29). Завершается часть IV обсуждением серийных средств, с помощью которых разрабатывают и управляют требованиями (глава 30). Заметьте, что в проекте может осуществляться управление определенными аспектами требований и одновременно выполняться разработка в других частях требований к продукту.

Процесс управления требованиями

Под управлением требованиями подразумевают все действия, по обеспечению целостности, точности и своевременности обновления соглашения о требованиях в ходе проекта. На рис. 27-1 показаны базовые операции по управлению требованиями в четырех основных категориях: управление версиями, управление изменениями, отслеживание состояния требований и отслеживание связей требований.

Рис. 27-1. Основные составляющие управления требованиями

В организации необходимо определить действия, которые, как ожидается, проектная команда будет выполнять для управления требованиями. Задокументируйте эти действия и организуйте тренинг для специалистов, чтобы сотрудники организации могли выполнять их последовательно и эффективно. Обратите внимание на следующее:

- на инструменты, приемы и соглашения для различения версий отдельных требований и их наборов;
- на составление наборов требований и их одобрение, а также на определение базовых версий требований (см. главу 2);
- на способы, с помощью которых новые требования и изменения существующих требований предлагаются, обрабатываются, обсуждаются и доводятся до всех заинтересованных лиц;
- на оценку влияния предложенного изменения;
- на атрибуты требований и процедуры отслеживания состояния требований, в том числе состояния требований, которые будут использоваться, и кто их будет менять;
- кто отвечает за обновление информации связей требований и когда это должно делаться;

- как отлеживаются и разрешаются проблемы с требованиями;
- как на планах и обязательствах проекта отразятся изменения требований;
- как эффективно использовать инструменты управления требованиями.

Вы можете включить всю эту информацию в одно описание процесса управления требованиями. Или же разработать отдельные процедуры для управления версиями, управления изменениями, анализа влияния и отслеживания состояния. Эти процедуры следует довести до сведения каждого сотрудника организации, поскольку они представляют общие функции, которые должны выполняться каждым в проектной команде. Подробнее о полезных инструментах управления требованиями см. главу 31.

Описания процесса должны определять роли в команде, которые отвечают за каждое из действий по управлению требованиями. Бизнес-аналитик проекта как правило, несет основную ответственность за управление требованиями. Он выбирает механизм хранения требований, определяет атрибуты требований, координирует обновление данных о состоянии и отслеживает обновление данных, а также следит за изменениями в операциях, если это необходимо. Описание процесса также должно определять ответственного за изменения в процессе управления требованиями, порядок работы с исключениями и путь передачи вопросов вверх по цепочке ответственности в случае возникновения проблем.

> **Внимание!** Если никто из участников проекта не несет ответственности за выполнение этапов управления требованиями, не следует ожидать их выполнения. Аналогично, если отвечают «все», то каждый человек может ожидать, что кто-то другой выполнит те или иные действия, поэтому они могут легко остаться невыполненными.

Базовое соглашение о требованиях

Стадия разработки подразумевает сбор информации, анализ, уточнение и утверждение требований к ПО в проекте. К результатам разработки требований относятся бизнес-требования, пользовательские требования, функциональные и нефункциональные требования, словарь данных и различные модели анализа. После рецензирования и утверждения любой из определенных наборов этих элементов составляет базовую версию требований. Как говорилось в главе 2, *базовая версия* (baseline) — это набор требований, согласованный заинтересованными лицами и часто определяющий содержимое определенного запланированного выпуска или итерации разработки. В проекте могут быть дополнительные соглашения относительно состава результатов, ограничений, графиков, бюджетов, переходных требований и контрактов, но это выходит за рамки этой книги.

Принятая базовая версия требований — как правило, версия становится базовой после официальных рецензирования и утверждения — передается в систему управления конфигурацией (или изменениями). Последующие

изменения разрешается вносить в нее только через определенный в проекте процесс управления изменениями. До принятия базовой версии требования все еще изменяются, поэтому не имеет смысла ограничивать модификацию какими-то процедурами. Базовая версия может включать часть или все требования определенной спецификации (для всего продукта или одного выпуска) или определенный набор требований, хранящийся в системе управления требованиями, или согласованный набор пользовательских историй для одной итерации проекта гибкой разработки (agile).

Если границы выпуска меняются, в соответствии с этим нужно обновлять базовую версию требований. Необходимо ясно отличать требование в базовой версии от других, которые были предложены, но не утверждены, назначены в другую базовую версию или остаются не назначенными в резерве продукта. Если требования определены в форме документа, такого как спецификация требований к ПО, он должен быть ясно определен как базовый, чтобы отличать его от серии выполненных ранее набросков версий, которые еще не утверждены. Хранение требования в средстве управления требованиями облегчает идентификацию требований, принадлежащих определенной базовой версии, и управление изменениями в ней.

Разработчики, принимающие предлагаемые изменения или добавления требований, не всегда способны соблюдать текущий график и выполнять обязательства, касающиеся качества. Менеджер проекта должен обговорить изменения этих обязательств с другими менеджерами, работающими над проектом, клиентами и всеми заинтересованными в проекте лицами. При введении новых или изменении существующих требований существует несколько вариантов корректировки проекта:

- откладывание или полный отказ от реализации требований с низкими приоритетами;
- привлечение дополнительных сотрудников или аутсорсинг части работы;
- продление графика или добавление дополнительных итераций в проект гибкой разработки;
- снижение качества с тем, чтобы поставить продукт к оговоренному сроку.

Ни один из этих способов нельзя назвать идеальным для всех случаев, поскольку гибкость проектов зависит от функций, опыта специалистов, бюджета, графика и качества (Wiegers, 1996). При принятии решений полагайтесь на бизнес-цели проекта и приоритеты, установленные при планировании проекта ключевыми фигурами из тех, кто заинтересован в проекте. Независимо от того, какова будет ваша реакция на корректировку требований, при необходимости примите как данность корректировку ожиданий и обязательств. Это гораздо лучше, чем надеяться, что каким-то образом к принятой в начале проекта дате выпуска все новые функции появятся в продукте, причем без таких последствий, как перерасход бюджета, переутомление команды или снижение качества.

Управление версиями требований

Управление версиями — идентификация уникальным образом разных версий каждого элемента — применяется к отдельным требованиям и их наборам, которые обычно представлены в форме документа. Начинайте контролировать версии сразу после того, как сделаете предварительный набросок этого документа, чтобы отслеживать всю историю изменений.

Каждая версия требований должна уникальным образом идентифицироваться. У каждого члена команды должен быть доступ к текущей версии требований, а изменения необходимо ясно документировать и доводить до всех заинтересованных лиц. Чтобы минимизировать путаницу и непонимание, назначьте право обновления требований строго определенным лицам и убедитесь, что идентификатор версии изменяется при каждом изменении требования. Каждая версия документа требований или каждое требование должно содержать историю изменений, где указываются внесенные изменения, дата каждого из них, лицо, внесшее изменение, а также причина изменения.

Это не дефект, а функция!

Разработчики, работающие по контракту, как-то получили массу отчетов об ошибках от людей, тестировавших последнюю версию системы, которую они только что поставили клиенту. Разработчики были в недоумении — система успешно прошла все их собственные проверки. После обстоятельного изучения проблемы, выяснилось, что клиенты тестировали новое ПО, сверяясь с устаревшей спецификацией требований. То, что тестировщики в отчетах называли ошибками, на самом деле было функциями. В обычной ситуации это просто шутка, из тех, что любят программисты. Тестировщикам понадобилось много времени, чтобы переписать тесты в соответствии с последней версией спецификации и повторно протестировать продукт, а все из-за отсутствия управления версиями. Другой коллега, столкнувшийся с аналогичным непониманием при тестировании из-за отсутствия информации об изменении, рассказал: «Мы потратили на исправление от четырех до шести часов, которые пришлось потратить без возмещения, потому что мы не могли выставить счет за эти часы. Я думаю, что специалисты по разработке ПО придут в ужас, если умножат такие потраченные впустую часы на почасовую ставку и увидят, сколько денег они в виде недополученного дохода они теряют».

Аналогичный конфуз может произойти, когда над проектом работает несколько бизнес-аналитиков. Один бизнес-аналитик может начать редактировать версию 1.2 спецификации требований к ПО, а несколькими днями позже другой бизнес-аналитик может поработать над теми же требованиями, назначив им ту же версию 1.2, даже не подозревая о конфликте версий. Очень скоро изменения теряются, требования перестают быть актуальными, работу приходится переделывать и углубляется недопонимание.

Наиболее надежный метод управления версиями заключается в хранении требований в базе данных серийного средства управления требованиями, как описано в главе 30. Эти средства отслеживают полную историю изменений требований, что особенно важно, если нужно вернуться к более ранней версии требования. Такое средство позволяет вносить комментарии, где можно разумно обосновать решение о добавлении, изменении или удалении требования; эти комментарии могут оказаться полезными при обсуждении требования в будущем.

Если вы сохраняете требования в документах, вы можете отследить коррективы с помощью режима изменений текста. Эта функция выделяет внесенную правку, зачеркивая удаленные фрагменты, подчеркивая добавленные и помечая вертикальными линиями на полях положение каждого изменения. После того как вы составите базовую версию документа с такими пометками, сначала создайте архив версии с пометками, затем примите все коррективы, а затем сохраните очищенную версию за новую базовую версию для следующего раунда изменений. Храните документы требований в таком средстве управления версиями, как тот, что применяется для контроля исходного кода с помощью системы управления версиями (check-in и check-out). Это позволит при необходимости вернуться к более ранним версиям и знать, кто, когда и почему изменил документ. (Кстати именно так мы писали эту книгу. Мы писали главы в Microsoft Word, используя режим исправлений в процессе работы над главами. Несколько раз нам приходилось возвращаться к предыдущим версиям.)

Я знаю об одном проекте, когда несколько сотен написанных в Microsoft Word вариантов использования продукта сохраняется в таком средстве управления версиями. Члены команды имеют доступ ко всем предыдущим версиям каждого варианта использования, для которых фиксируется история изменений. Бизнес-аналитику этого проекта и его заместителю, отвечающему за архивирование, предоставлен доступ на чтение и запись; остальным членам команды — доступ только на чтение. Такой подход хорошо себя зарекомендовал в этой команде.

Простейший механизм управления версиями — именовать вручную каждую версию спецификации требований к ПО согласно стандартному соглашению. Попытка различать версии документов по дате изменения или дате печати часто приводит к ошибкам и неразберихе. Несколько команд, в которых я работал, успешно применяли такое соглашение: первая версия любого нового документа называется «Версия 1.0 черновик 1». Следующий черновик называется «Версия 1.0 черновик 2». Автор последовательно увеличивает номер черновика при каждом изменении до тех пор, пока документ не будет одобрен или утверждена базовая версия документа. Тогда название меняется на «Версия 1.0 утвержденная». Следующие версии называются «Версия 1.1 черновик 1» при небольшом изменении или «Версия 2.0 черновик 1» при значительном изменении. (Разумеется, термины «небольшой» и «значительный» субъективны или, по крайней мере, зависят от контекста.)

При применении этой схемы ясно различаются черновики и версии базового документа, однако необходимо соблюдать строгий ручной порядок в ведении документации.

Атрибуты требований

Представьте себе каждое требование в виде объекта, который обладает свойствами, отличающими его от других требований. В дополнение к тестовому описанию, каждое функциональное требование должно иметь несколько поддерживающих его фрагментов информации, или *атрибутов* (attributes), связанных с ним. Эти атрибуты представляют контекст и основу каждого требования, они располагаются за описанием предполагаемой функциональности. Вы можете сохранить атрибуты в таблице, базе данных или, что наиболее эффективно, в средстве управления требованиями. Сложно управлять более десятком атрибутов, когда они хранятся только в документах.

Средства управления требованиями предоставляют несколько сгенерированных системой атрибутов, а также предоставляют возможность определить другие атрибуты, часть из которых может создаваться автоматически. Эти средства позволяют фильтровать, сортировать выбранные подмножества требований на основании значений их атрибутов и запрашивать базу данных для их просмотра. Например, можно вызвать список всех высокоприоритетных требований, которые Шери должна реализовать в версии 2.3 и которые имеют статус Approved (Одобрено). Далее приведены возможные атрибуты требований:

- дата создания требования;
- номер текущей версии требования;
- автор, создавший требование;
- приоритет;
- состояние требования;
- происхождение или источник требования;
- логическое обоснование требования;
- номер выпуска или итерации, на которую назначено требование;
- контактное лицо или ответственный за принятие решений по внесению изменений в требование;
- метод проверки или критерий приемки.

Для чего нужно это требование?

Менеджер продукта в компании, выпускающей электронные измерительные приборы, хотел просто записать включенные требования, потому что продукт конкурентов обладал теми же функциями. Такие функции хорошо отмечать с помощью атрибута Rationale (Обоснование), позволяющего указать, почему конкретное требование включено в продукт. Представьте,

что вы включили определенное требование, потому что оно отвечает потребностям определенной группы пользователей. В дальнейшем в отделе маркетинга решили, что эта группа нас больше не интересует. Наличие такого обоснования в виде атрибута требования поможет решить, стоит ли реализовывать такое требование.

Еще один бизнес-аналитик удивляется включению требований без явных на то оснований. Он сказал: «По своему опыту могу сказать, что многие требования не подтверждены реальной потребностью. Они введены, потому что клиенты не понимают технологий, некоторые заинтересованные лица очарованы технологией и хотят произвести впечатление или наш отдел продаж осознанно или по недомыслию ввел клиента в заблуждение». Если не представить убедительного обоснования и не связать требование с бизнес-потребностью, бизнес-аналитик может поставить под сомнение, стоит ли вообще тратить на него силы.

Внимание! Слишком большое число требований может обескуражить команду. Члены команды просто не станут задавать значения всех атрибутов во всех требованиях и не будут эффективно использовать информацию атрибутов. Начните с трех-четырех ключевых атрибутов. В дальнейшем добавляйте атрибуты, только если уверены, что это принесет реальную пользу.

Набор требований, запланированный для определенной версии, будет изменяться по мере добавления новых и удаления или отсрочкой до более поздних версий существующих требований. Разработчики могут перемещать отдельные документы требований между выпусками и итерациями. Если отложенные требования или те, от которых вы решили отказаться, остаются в спецификации требований, читатели спецификации могут запутаться в том, какие требования включены в конкретную базовую версию. Один из способов решения этой проблемы — сохранить требования в средстве управления требования и определить атрибут Release Number (Номер версии). Отсрочка требования означает изменение его запланированного выпуска, поэтому просто обновив номер версии, вы передвинете требование в другую базовую версию. Теми требованиями, от которых вы решили отказаться, лучше всего управлять с помощью атрибута состояния, как описано в следующем разделе.

На определение и обновление этих атрибутов требуется часть затрат, отведенных на управление требованиями, однако эти инвестиции в значительной степени окупаемы. В одной компании периодически генерировался отчет о требованиях, в котором показывалось, какое из 750 требований в трех связанных спецификациях назначено каждому дизайнеру. Один из дизайнеров обнаружил несколько требований, которые он не считал своими. По его расчетам он сэкономил от одного до двух месяцев, отведенных на переделку дизайна, которые бы понадобились, если бы он выяснил ситуацию с этими требованиями позднее. Чем крупнее проект, тем легче возникают недопонимания, отнимающие лишнее время.

Отслеживание состояния требований

«Как движется работа над той подсистемой, Иветт», — спросил менеджер проекта Дейв.

«Довольно хорошо, Дейв. Готово около 90%».

Дейв был озадачен: «А разве пару недель назад ты не сделала примерно 90%?»

Иветт ответила: «Я думала, что да, но теперь эти 90% действительно готовы».

Иногда разработчики ПО слишком оптимистичны, когда сообщают об объеме выполненных работ по задаче. Обычный синдром «90 процентов» не говорит Дейву, насколько Иветт на самом деле приблизилась к завершению своей подсистемы. Но Иветт может сообщить о состоянии дел и так: «Все очень даже неплохо, Дейв. В моей подсистеме 84 требования: 61 из них реализованы и проверены, 14 реализованы, но не проверены и 9 требований ожидают реализации». Отслеживание состояния каждого функционального требования на протяжении всей разработки позволяет более точно оценивать готовность проекта.

Состояние — один из атрибутов требований, предложенный в предыдущем разделе. Отслеживание состояния означает определение готовности по отношению к тому, что это означает в текущем цикле разработки. Например, вы планируете реализовать в следующей версии только часть варианта использования, оставив полную реализацию до будущих версий. В этом случае вам нужно отлеживать состояние тех функциональных требований, которые запланированы для ближайшего выпуска, потому что этот набор требований должен быть выполнен на 100% до выпуска продукта.

> **Внимание!** Существует старая шутка, что на первую половину проекта по разработке тратится 90% ресурсов, а на остальную половину — оставшиеся 90% ресурсов. Чересчур оптимистичная оценка и попустительское отношение к отслеживанию состояния — верный путь к перерасходам в проекте.

В таблице 27-1 перечислено несколько возможных состояний требований. Некоторые специалисты добавляют другие состояния, такие как состояние Designed (Разработано) (если элементы дизайна, в которых отражены функциональные требования, созданы и проверены) и Delivered (Выпущено) (ПО отдано в руки пользователей для приемки или бета-тестирования). Полезно отслеживать требования, от которых вы отказались, и причины этого, потому что отвергнутые требования имеют обыкновение «всплывать» в ходе разработки или в последующих проектах. Состояние Rejected (Отклонено) позволяет оставить предложенное требование доступным для будущего использования, не создавая беспорядка в наборе утвержденных требований для определенного выпуска. Не обязательно отлеживать все возможные состояния в табл. 27-1 — выберите те, что важны для работы с требованиями.

Табл. 27-1. Рекомендуемые состояния требований

Состояние	Определение
Proposed (Предложено)	Требование запрошено уполномоченным лицом
In Progress (Разработка)	Бизнес-аналитик активно работает над требованием
Drafted (Подготовлено)	Написана начальная версия требования
Approved (Одобрено)	Требование проанализировано, его влияние на проект просчитано, и оно было размещено в базовой версии определенной версии. Ключевые заинтересованные в проекте лица согласились с этим требованием, а разработчики ПО обязались реализовать его
Implemented (Реализовано)	Код, реализующий требование, разработан, написан и протестирован. Определена связь требования с соответствующими элементами дизайна и кода. ПО, реализующее требование теперь готово для тестирования, рецензирования и другой проверки
Verified (Проверено)	Требование удовлетворяет критериями приемки, то есть подтверждено корректное функционирование реализованного требования. Определена связь требования с соответствующими вариантами тестирования. Теперь требование считается завершенным
Deferred (Отложено)	Одобренное требование теперь запланировано для реализации в более позднем выпуске
Deleted (Удалено)	Утвержденное требование удалено из базовой версии. Опишите причины удаления и назовите того, кто принял это решение
Rejected (Отклонено)	Требование предложено, но не запланировано для реализации ни в одном из будущих выпусков. Укажите причины отклонения требования и того, кто принял это решение

Распределение требований по нескольким категориям состояния имеет больший смысл, чем попытка контролировать процент завершения каждого требования или всей базовой версии. Обновляйте состояние требования, только когда удовлетворены определенные условия перехода. Определенные изменения состояния также ведут к обновлению данных о связях требований, когда указывается, в каких элементах дизайна, кода и тестирования отражено требование, как показано в табл. 29-1 в главе 29.

На рис. 27-2 показано, как контролировать состояние требования в гипотетическом 10-месячном проекте: на графике показано, сколько процентов требований к системе в каждом состоянии имеется в конце каждого месяца. Обратите внимание, что при этом не устанавливается, изменяется ли со временем количество требований в базовой версии. Число требований увеличи-

вается при расширении границ проекта и уменьшается при удалении функциональности из базовой версии. Кривые иллюстрируют, как достигается такая цель проекта, как полная проверка всех утвержденных требований. Основная часть работы считается выполненной, когда всем требованиям назначено состояние Verified (Проверено), Deleted (Удалено) или Deferred (Отложено).

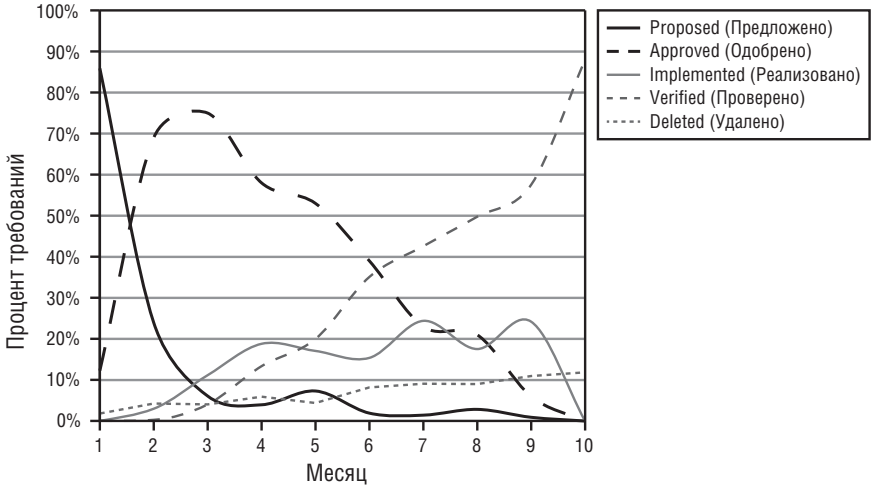


Рис. 27-2. Отслеживание распределения состояний требований в цикле разработки проекта

Разрешение проблем с требованиями

В процессе проекта возникнет много связанных с требованиями проблем, решений и конфликтов. Это могут быть элементы, помеченные как «TBD» (to be determined — необходимо определить), отложенные решения, необходимая информация, неразрешенные конфликты и т.п. Очень легко потерять эти проблемы из виду. Регистрируйте проблемы в средстве отслеживания дефектов, чтобы у всех заинтересованных лиц был доступ к ним. Сделайте процесс отслеживания и разрешения проблем максимально простым, чтобы ничего не оставалось без внимания. У использования средства отслеживания дефектов есть ряд преимуществ:

- проблемы, обнаруженные в ходе разных процессов рецензирования требований, собираются вместе и ни одна из них не теряется;
- менеджер проекта может легко увидеть текущее состояние всех проблем;
- каждой проблеме можно назначить отдельного ответственного;
- сохраняется история обсуждения проблемы;
- команда может начать разработку с известным набором открытых проблем, не ожидая завершения работы над спецификацией требований.

Проблемы с требованиями нужно разрешать своевременно, чтобы они не мешали созданию базовой версии высококачественных требований для

следующего выпуска или итерации. График сгорания задач, показывающий оставшиеся проблемы и скорость их разрешения, помогает спрогнозировать, когда все проблемы будут закрыты, чтобы при необходимости ускорить их разрешение. (Пример графика сгорания задач см. в разделе «Управление требованиями в проектах гибкой разработки» дальше в этой главе.) Разбиение проблем на категории помогает определить разделы требований, нуждающихся в дополнительной работе. Несколько открытых проблем в разделе могут означать, что требование еще не прошло рецензирование или открытые проблемы в целом разрешены.

Практически все дефекты, зарегистрированные на ранних стадиях проекта, связаны с проблемами в требованиях — в их числе запросы на уточнение требований, решения о границах, вопросы возможности реализации и необходимость доработки в самих требованиях. В процессе рецензирования требований все участники могут вносить вопросы. В табл. 27-2 перечислены обычные проблемы, которые возникают с требованиями.

Табл. 27-2. Проблемы, которые возникают с требованиями

Тип проблемы	Описание
Вопрос по требованию	Что-то, что непонятно или не решено в отношении требования
Отсутствующее требование	В процессе реализации разработчики обнаружили отсутствующее требование
Неправильное требование	Требование неверно. Его нужно исправить или удалить
Вопрос по реализации	В процессе реализации у разработчиков возникают вопросы о том, как что-то должно работать или о вариантах дизайна
Дублирование требования	Обнаружены два или больше эквивалентных требований. Надо оставить одно, а остальные удалить
Ненужное требование	Требование просто больше не нужно

Могут произойти очень нехорошие вещи, если у вас не упорядочен процесс разрешения проблем с требованиями. В одном проекте один участник говорил, что нам нужно обрабатывать что-то «в портале». Я впервые услышал о портале и спросил об этом. Участник проекта заверил меня, что серийное решение, которое планируется приобрести, включает портал, который просто нужно будет соответствующим образом настроить. Мы не включали в свои планы никаких требований о наличии портала, поэтому я подумал, что здесь у нас пробел. Я попросил коллегу зафиксировать проблему с порталом, чтобы мы не упустили эту потребность. Несколькими неделями позже мне пришлось покинуть проект.

Как оказалось, мой коллега записал проблему с порталом на доске, которую потом стерли, так что он не зафиксировал ее в средстве отслеживания

дефектов. Через полгода после начала проекта ко мне пришел руководитель заказчика, взбешенный тем, что никто не выяснил, что требуется портал. Мне пришлось выяснять, почему мы не разработали требования к порталу: мы просто забыли о нем. Регистрация проблемы в средстве отслеживания дефектов избавила бы нас от горячечной доработки в последний момент и неудовольствия клиента.

Определение усилий, необходимых для управления требованиями

Как и при разработке требований, в план проекта надо включить задачи и ресурсы для выполнения задач по управлению требованиями, описанные в этой главе. Если вы выясните, сколько усилий тратится на управление требованиями, вы сможете оценить — мало их, достаточно или слишком много — и изменить рабочие процессы и будущие планы соответствующим образом. Карл Вигерс (Wiegers, 2006) изучает измерение различных аспектов работы над требованиями в проекте.

Для измерения непосредственных усилий на разработку и управление проектом необходимо заняться корпоративной культурой и дисциплиной сотрудников — только так удастся фиксировать ежедневную работу (Wiegers, 1996a). На это уходит не так много времени. Команда сможет получить ценные идеи, поняв, как *на самом деле* тратятся усилия на различные задачи проекта по сравнению с *личным представлением* о потраченном времени и тем, как должно тратиться время. Отслеживание усилий также указывает, справляется ли команда с запланированными действиями по работе с требованиями.

Заметьте, что рабочие усилия — это не то же самое, что прошедшее календарное время. Выполнение задач может прерываться или необходимо переговорить с другими людьми, что ведет к задержкам. Общие усилия, необходимые для выполнения задачи и измеряемые в человеко-часах, не обязательно изменятся из-за этих факторов (хотя частые прерывания снижают продуктивность отдельных людей), однако продолжительность выполнения задачи увеличится.

В процессе отслеживания усилий по разработке требований может оказаться полезным отделить время, затрачиваемое бизнес-аналитикам, от времени работы других участников проекта. Отслеживание времени бизнес-аналитиков поможет вам планировать их усилия в будущих проектах (подробнее об оценке времени бизнес-аналитика см. главу 19). Измерение общих усилий, потраченных на разработку требований всеми заинтересованными лицами, дает представление об общей стоимости работы над требованиями в проекте. Зафиксируйте число часов, потраченных на разработку требований, в том числе на:

- планирование работы с требованиями в проекте;
- проведение семинаров и интервью, анализ документов и выполнение других действий по выявлению требований;
- написание спецификаций требований, создание моделей анализа и определение приоритетов требований;
- создание и оценку прототипов в процессе разработки требований;
- рецензирование требований и выполнение других действий по проверке требований.

Считайте усилия для выполнения следующих действий усилиями по управлению требованиями:

- конфигурирование средства управления требованиями в проекте;
- предложение изменения требований и новых требований;
- оценка предложенных изменений, включая оценку влияния изменения и принятие решений;
- обновление хранилища требований;
- сообщение об изменениях требований заинтересованным группам и отдельными заинтересованными лицами;
- контроль и отчет о состоянии требований;
- сбор информации о связях требований.

Время, затраченное на преобразование требований в дизайн, — это не затраты, а отличная инвестиция в успех проекта. Чтобы оценить их пользу, сравните эти затраты времени с временем, которое команда тратит на разрешение проблем из-за того, что некоторые вещи *не были* сделаны, — это цена, которую приходится платить из-за плохого качества.

Управление требованиями в проектах гибкой разработки

В проектах гибкой разработки (agile) работа с изменениями ведется путем построения продукта в рамках серии итераций и управления динамическим резервом (backlog) продукта, то есть работы, которую нужно выполнить. Как описано в главе 2, заинтересованные лица согласовывают истории, которые должны реализовываться в каждой итерации. Новые истории, добавляемые пользователями во время итерации, приоритизируются в совокупности с резервом проекта и назначаются на будущие итерации. Новые истории могут вытеснить низкоприоритетные истории, если команда не хочет менять график выпуска. Как и во всех других проектах, задача всегда заключается в работе над наиболее приоритетными историями, чтобы максимально быстро предоставить клиентам полезный продукт. Подробнее об управлении изменениями требований в проектах гибкой разработки см. главу 28.

В некоторых командах, особенно в крупных или распределенных, используется средство управления проектом гибкой разработки для отслеживания

состояния итерации и назначенных на нее историй. Истории и соответствующие им приемочные критерии и тесты можно все разместить в резерве продукта или средстве управления пользовательскими историями. Состояние истории можно отслеживать с использованием состояний, аналогичным тем, которые описаны ранее в табл. 27-1 (Leffingwell, 2011):

- In backlog (В резерве) (история пока не назначена на конкретную итерацию);
- Defined (Определена) (подробности истории обсуждены и поняты, и написаны приемочные тесты);
- In Progress (Разработка) (история в процессе реализации);
- Completed (Завершено) (история полностью реализована);
- Accepted (Принята) (пройдены приемочные тесты);
- Blocked (Блокирована) (разработчик не может продолжить разработку, пока не будут разрешены другие вопросы).

В проектах гибкой разработки продвижение проекта отслеживается с помощью графика сгорания задач (Cohn, 2004; Cohn, 2005). В команде общий объем работы, которую надо выполнить в проекте, часто измеряют в баллах историй, которые определяются в процессе анализа пользовательских историй в резерве продукта (Cohn, 2005; Leffingwell, 2011). Таким образом, общее число баллов историй пропорционально усилиям, которые команда должна потратить на реализацию этих требований. Конкретные пользовательские истории назначаются на итерации на основе их приоритетов и размера в баллах. Прошлая или средняя скорость разработки определят число баллов историй, которые команда сможет реализовать в итерации или в конкретный календарный срок.

Остающиеся в конце итерации баллы историй в резерве отмечаются на графике. Эта общая цифра изменяется по мере завершения работы, лучшего понимания и переоценки текущих историй, добавления новых историй и удаления завершенной работы из резерва. Таким образом, график сгорания задач показывает общий объем остающейся работы на определенный момент времени, а не отслеживает число и состояние отдельных функциональных требований или функций (размер которых может быть разным).

На рис. 27-3 показан график сгорания задач гипотетического проекта. Заметьте, что измеренный в баллах историй объем остающейся работы, на самом деле увеличился в итерациях 2, 3 и 5. Это говорит, что в резерв добавилось больше новой функциональности, чем было завершено или удалено в течение итерации. График сгорания задач помогает избежать синдрома «90 процентов», наглядно показывая объем остающейся, а не завершенной работы, которая не отражает неизбежное увеличение границ проекта. Наклон графика сгорания задач также отражает прогнозируемую конечную дату проекта — точку, когда резерв будет пуст.

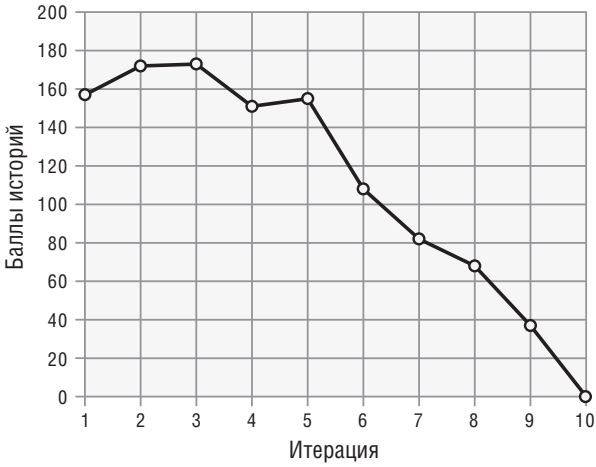


Рис. 27-3. Пример графика сгорания задач в проекте гибкой разработки

Почему нужно управлять требованиями

Управление требованиями жизненно необходимо независимо от типа жизненного цикла проекта — будь то водопадная модель или модель гибкой разработки. Управление требованиями помогает убедиться, что усилия, затраченные на разработку требований, не потрачены впустую. Эффективное управление требованиями может уменьшить неверные ожидания: заинтересованные лица будут проинформированы о текущем состоянии требований в ходе процесса разработки. Это позволяет понимать, куда движется проект, как идут дела и когда проект прибудет в конечную точку.

Что дальше?

- Задокументируйте процессы, которым будет следовать ваша организация для управления требованиями в каждом проекте. Попросите нескольких аналитиков нарисовать, просмотреть, попробовать выполнить и одобрить действия и результаты. Этапы процесса, которые вы определите, должны быть практичными и выполнимыми, и они должны добавлять определенную ценность проекту.
- Если вы не используете средство управления требованиями, определите схему управления версиями документации по требованиям. Научите бизнес-аналитиков применять эту схему.
- Выберите состояния, которые вы хотите использовать для описания жизненного цикла функциональных требований или пользовательских историй. Нарисуйте диаграмму перехода состояний, чтобы показать условия, вызывающие изменения состояний.
- Определите текущее состояние для каждого функционального требования в базовой версии. Обновляйте состояние по мере разработки.

Глава 28

Изменения случаются

«Как продвигается разработка, Гленн?» — спросил Дейв, менеджер проекта Chemical Tracking System на совещании, посвященном состоянию проекта.

«Я не так далеко продвинулся, как рассчитывал, — признался Гленн. — Сейчас я добавляю новую функцию запроса в каталоге для Харуми, и это требует намного больше времени, чем я предполагал».

Дейв был озадачен. «Я не помню, чтобы мы обсуждали новую функцию запроса каталога на недавней встрече совета по управлению изменениями. Харуми подала этот запрос через процесс управления изменениями?»

«Нет, она обратилась ко мне напрямую с этим предложением, — сказал Гленн. — Мне следовало попросить ее оформить его, как официальный запрос на изменение, но все выглядело настолько просто, что я сказал ей, что поработаю над этим. Выяснилось, что не все так просто. Каждый раз, когда мне кажется, что закончил, я понимаю, что я пропустил изменение, которое должно быть в другом файле, поэтому мне приходится исправлять это, собирать компонент и заново тестировать его. Я думал, мне хватит шести часов, однако я уже потратил на это три дня. Знаю, что задерживаю следующую сборку. Мне закончить с этой функцией или заняться тем, над чем я работал раньше?»

Большинство разработчиков сталкивались с простыми на первый взгляд изменениями, которые оказывались более сложными, чем они ожидали. Разработчики иногда не выполняют — или не могут выполнить — реалистичные расчеты затрат и других последствий предложенного изменения ПО. И когда разработчик, который хочет быть любезным, соглашается добавить улучшение, запрашиваемое пользователем, требования изменяются через «черный ход», а не утверждаются соответствующими заинтересованными лицами. Такие неконтролируемые изменения — повсеместный источник хаоса в проекте, нарушения графика, проблем с качеством и порчи отношений. В этой главе описываются как формальные приемы управления изменениями, так управление изменениями в проектах гибкой разработки.

Почему нужно управлять требованиями

Изменение ПО само по себе неплохое дело. Определить все требования к продукту практически невозможно, а по ходу проекта изменяются и сопутствующие обстоятельства: возникают новые возможности на рынке, меняются правила и политики, а также развиваются бизнес-потребности. Эффективная команда разработчиков ПО быстро отреагирует на необходимые изменения, чтобы продукт, который они создают, был своевременно поставлен клиентам. Организация, серьезно относящаяся к управлению своими проектами по разработке ПО, должна убедиться, что:

- предложенные изменения требований тщательно оценены до их ввода в дело;
- надлежащие лица принимают бизнес-решения, будучи хорошо информированными о запрошенных изменениях;
- одобренные изменения доведены до сведения всех заинтересованных лиц;
- одобренные изменения требований доведены до всех заинтересованных лиц;
- изменения требований добавляются в проект единообразным и эффективным образом.

Тем не менее у изменений всегда есть цена. Простая веб-страница корректируется быстро и легко; новшества же в дизайне интегрированной микросхемы может обойтись в десятки тысяч долларов. Даже на отклонение запроса на изменение тратятся ресурсы, необходимые для подачи, оценки и принятия решения. Если заинтересованные в проекте лица не управляют изменениями в ходе разработки, им неизвестно, что именно будет получено в результате, а это неизбежно ведет к расхождениям в ожиданиях.

Если разработчик реализует изменение требования напрямую в коде, не общаясь с другими членами команды, возможны проблемы. Из-за этого задокументированное требование становится неточным представлением возможностей продукта. Код может стать неустойчивым и хрупким, если изменения вносятся без учета архитектуры и дизайна. В одном проекте разработчики ввели новую и изменили существующую функциональность, о чем остальные члены команды не знали до тестирования системы. При этом понадобилась незапланированная переработка процедур тестирования и пользовательской документации. Последовательные приемы управления изменениями помогают предотвратить такие проблемы, а также связанные с этим расстройства, переделки и трату времени.

Бойтесь изменений в обход

Поставщик и клиент однажды создали огромную путаницу в проекте, когда пропустили процесс изменений в контрактном проекте. Поставщик (отобранный ИТ-отделом, но нанятый бизнес-подразделением) должен был разработать новое приложение мобильной рабочей станции. Требо-

вания были собраны группой из 10 специалистов предметной области. После этого ведущий клиент из бизнес-подразделения решил, что нужно еще внести изменения в требования. Не веря, что исправления получат поддержку, он сговорился с разработчиками поставщика обойти согласованные требования. Они арендовали комнату в отеле и работали в секрете, внося изменения в код на лету. Когда тестировщики обнаружили, что результаты не соответствовали требованиям, заговор вскрылся. Анализ изменений и ожидаемых результатов задним числом вылилось организации в значительный объем дополнительного времени и усилий.

По странной иронии судьбы ведущий клиент позже стал бизнес-аналитиком. Он принес свои самые глубокие извинения, потому что только после этого понял, насколько его действия нарушили работу всей команды.

Управление незапланированным ростом объема

В идеальном мире вам удастся собрать все требования к новой системе до начала сборки, и они останутся стабильными в течение всей разработки. Это основа последовательной или водопадной модели разработки ПО, но на практике так не бывает. На определенном этапе вам необходимо «заморозить» требования до определенной версии или вам никогда не удастся получить результат. Однако, если вы приостановите работу над требованиями слишком рано, то вы проигнорируете то обстоятельство, что клиенты не всегда уверены в том, что именно им нужно; клиентам понадобились изменения, и разработчикам необходимо отреагировать на это.

Незапланированным изменением требований считается предложение новой функциональности и существенной модификации после утверждения базовой версии требований к проекту (см. главу 2). Чем дольше продолжается работа над проектами, тем больше растет их объем. Требования к системе управления информацией, как правило, увеличиваются на 1-3% в месяц (Jones, 2006). Определенные изменения требований абсолютно правомочны, неизбежны и даже благоприятны. Но рост объема, при котором в проект постоянно включается новая функциональность без учета ресурсов, графика или целей качества, гораздо коварнее. Проблема не в самом изменении требований, а в том, что поздние изменения оказывают огромное влияние на уже выполненную работу. Если все изменения будут одобряться, может оказаться, что нет никакой надежды разработать продукт, — и он не будет разработан.

Первый шаг при управлении незапланированным ростом объема — документирование бизнес-целей, концепции, границ и ограничений новой системы, как описано в главе 5. Оценивайте каждое предложенное требование или функцию, соотносясь с бизнес-требованиями. Если задействовать клиентов в сборе информации, то снижается количество требований, упущенных из

внимания. Составление прототипов — это еще один эффективный прием управления незапланированным ростом объема. Прототип помогает разработчикам и пользователям выработать общее понимание нужд пользователей и необходимых решений. Короткие циклы разработки при инкрементальной разработке системы позволяют вносить изменения часто.

Наиболее эффективный прием для управления незапланированным ростом объема — это умение сказать: «Нет» (Weinberg, 1995). Люди в принципе не любят отказывать, а разработчиков могут прессинговать по поводу каждого предложенного требования. Такие принципы, как «Клиент всегда прав» и «Мы сделаем все, чтобы клиент остался доволен», звучат прекрасно в теории, однако за их выполнение нужно платить. Игнорируя необходимость платы, вы никак не измените тот факт, что коррективы не могут произойти просто так. Президент одной компании, специализирующейся на поставке программного обеспечения, привык слышать от менеджера по разработке в ответ на предложение новой функции: «Не сейчас». «Не сейчас», — звучит более привлекательно, чем простой отказ. В этом слышится обещание включить функцию в следующие версии.

> **Внимание!** «Заморозка» требований для новой системы слишком рано — после выполнения первоначального сбора информации — не умно и не практично. Лучше определите базовую версию, когда решите, что требования достаточно хорошо определены, чтобы начать разработку, а затем управляйте неизбежными изменениями, чтобы минимизировать их негативное влияние на проект.

Политика управления изменениями

Руководство должно ясно довести до сведения сотрудников политику, в которой описываются ожидания того, как участники проекта должны поступать с предложенными изменениями требований и другими важными артефактами проекта. Политики имеют смысл, только если они реалистичны, приносят пользу и проводятся в жизнь. Следующие положения политики управления изменениями могут быть полезными:

- все изменения требований должны вноситься в соответствии с процессом. Если запрос на изменение не подается в соответствии с этим процессом, он не рассматривается;
- нельзя заниматься дизайном или реализовывать неутвержденные изменения, кроме проверки их осуществимости;
- сам по себе запрос на изменение не гарантирует выполнение этого изменения. Совет по управлению изменениями проекта (change control board, CCB) решает, какие изменения следует реализовать;
- содержимое базы данных изменений должно быть видимым для всех заинтересованных в проекте лиц;
- анализ влияния изменения необходимо выполнять для каждого изменения;

- каждое включенное изменение требования должно отслеживаться вплоть до утверждения запроса на это изменение;
- обоснование каждого утверждения или отклонение запроса на изменение необходимо документировать.

Разумеется, мелкие изменения вряд ли повлияют на проект, а существенные воздействуют довольно сильно. На практике вы можете делегировать принятие определенных детальных решений, касающихся требований, разработчикам, однако ни одно решение, влияющее на работу более чем одного человека, не должно идти в обход процесса управления изменениями. При этом процесс должен предлагать «быстрый путь», чтобы ускорить не рискованные и дешевые изменения в сжатом цикле принятия решений.

Основные понятия процесса управления изменениями

Как-то оценивая процесс разработки ПО, я спросил специалистов, работающих над проектом, как они поступают с изменениями в требованиях к продукту. После неловкой паузы, один из них сказал: «Каждый раз, когда специалист отдела маркетинга хочет внести изменение, он обращается к Брюсу или Робин, потому что они всегда соглашаются, а мы нет». Это не показалось мне хорошим процессом внесения изменений.

Процесс управления изменениями позволяет руководителю проекта принимать информированное бизнес-решение, которое принесет максимум пользы клиенту и бизнесу, при этом контролируются затраты на жизненный цикл продукта и график проекта. Процесс позволяет отслеживать статус всех предложенных изменений и убедиться, что предложенные требования не были потеряны или упущены. После утверждения базовой версии набора требований, придерживайтесь этого порядка для внесения всех предлагаемых изменений в нее.

Клиенты и другие заинтересованные лица часто уклоняются от нового процесса, однако процесс управления изменениями — не препятствие для модификации. Это механизм суммирования и фильтрации, дающий уверенность, что в проект включены наиболее ценные изменения. Если предложенное изменение не настолько важно, чтобы заинтересованное в проекте лицо потратило пару минут на передачу его через стандартные, простые каналы, то стоит задуматься о его ценности вообще. Процесс управления должен быть тщательно задокументирован, максимально прост и, что важнее всего, эффективен.

> **Внимание!** Если вы предложите заинтересованным лицам новый процесс управления изменениями — неэффективный, громоздкий или слишком сложный, они постараются найти способ избежать этого — и, возможно, будут правы.

Управление изменениями требований схоже с процессом сбора и принятия решений по отчетам об ошибках. Эти же средства могут поддерживать

обе задачи. Однако следует помнить: инструмент не заменяет четко задокументированный процесс и ни одно из них не заменит надлежащего общения между заинтересованными лицами. Инструмент и зафиксированный на бумаге процесс нужно использовать как вспомогательные средства для критически важных обсуждений.

Если вам нужно внести изменение, начните с наивысшего уровня абстракции, затрагиваемого этим изменением, и далее переносите изменение на остальные компоненты системы через связанные с ним системные компоненты. Например, предложенное изменение может повлиять на вариант использования и его функциональные требования, однако никак не затронет бизнес-требования. Изменение требования к системе высокого уровня может воздействовать на множество требований к ПО и оборудованию во многих подсистемах. Некоторые изменения касаются только внутренних компонентов системы, например реализации коммуникационного сервиса. Они невидимы для пользователя и скорее относятся к изменениям дизайна или кода.

Описание процесса управления изменениями

На рис. 28-1 показан шаблон описания процесса управления изменениями для работы с модификациями требований. Пример процесса управления изменениями можно загрузить с веб-сайта этой книги. Если этот шаблон слишком сложен для вашей среды, упростите его для использования в менее формальных проектах.

1. Назначение и границы
2. Роли и ответственность
3. Состояние запроса на изменение
4. Входные критерии
5. Задачи
   - 5.1 Оценка запроса на изменение
   - 5.2 Принятие решения об изменении
   - 5.3 Реализация изменения
   - 5.4 Проверка изменения
6. Выходные критерии
7. Отчет о состоянии контроля изменений

Приложение. Атрибуты запросов на изменение

Рис. 28-1. Пример шаблона для описания процесса управления изменениями

Мы считаем полезным включить следующие четыре компонента во все процедуры и описания процесса:

- входной критерий — условия, которые должны быть удовлетворены до начала выполнения процесса;
- различные задачи, задействованные в процессе или процедуре, роли в проекте, отвечающие за выполнение каждой задачи, и другие лица, принимающие участие в выполнении задачи;

- последовательные действия для подтверждения того, что задачи были выполнены правильно;
- выходной критерий — условия, указывающие, когда процесс или процедура считаются успешно завершенными.

В оставшейся части этого раздела описываются различные разделы описания процесса управления изменениями.

1. Назначение и границы

Здесь описывается назначение процесса и определяются организационные границы, в которых он применяется. Также укажите, существуют ли какие-то специальные виды изменений, которые не подлежат контролю, например изменения в промежуточных продуктах, созданных в ходе проекта. Определите все термины, которые могут потребоваться для понимания остальной части документа.

2. Роли и ответственность

Перечислите роли членов команды, которые участвуют в процессе управления изменениями, и опишите их обязанности. В табл. 28-1 приведены некоторые типичные роли; адаптируйте их в соответствии с конкретной ситуацией. Один человек может выполнять несколько ролей. Например, председатель совета по управлению изменениями также может получать подаваемые запросы на изменения. В небольших проектах несколько, а возможно, и все роли выполняются одним человеком. Как сказал один опытный менеджер проектов: «Я считаю важным, чтобы представитель совета по управлению изменениями умел говорить о потребностях различных заинтересованных лиц, в том числе конечных пользователей, бизнес и сообщество разработчиков: нужно ли нам это, сможем ли мы продать это и как мы сможем это построить?»

Табл. 28-1. Возможные проектные роли в процессе управления изменениями

Роль	Описание и обязанности
Председатель совета по управлению изменениями	Возглавляет совет по управлению изменениями; как правило, обладает правом принятия окончательного решения, если совет не приходит к согласию; выбирает ответственных за оценку и внесение изменений
Совет по управлению изменениями	Группа, принимающая решение утвердить или отклонить каждое предложенное изменение для определенного проекта
Ответственный за оценку	Лицо, которое председатель совета просит проанализировать влияние предложенного изменения
Ответственный за внесение изменений	Лицо, отвечающее за внесение изменений в продукт, когда запрос на изменение утвержден
Инициатор	Лицо, подающее новый запрос на изменение

Табл. 28-1. (*окончание*)

Роль	Описание и обязанности
Получатель запросов	Лицо, которому передаются новые запросы на изменение
Контролер	Лицо, определяющее, корректно ли реализовано изменение

3. Состояние запроса на изменение

Запрос на изменение проходит через определенные состояния на протяжении своего жизненного цикла. Эти изменения состояния можно представить, используя диаграмму перехода состояний (см. главу 12), как показано на рис. 28-2. Обновляйте состояние запроса, только когда удовлетворяются определенные критерии. Например, состояние «Изменение внесено» можно задать после изменения всех связанных рабочих продуктов в процессе реализации изменения, будь то одно требование или набор связанных результатов разработки.

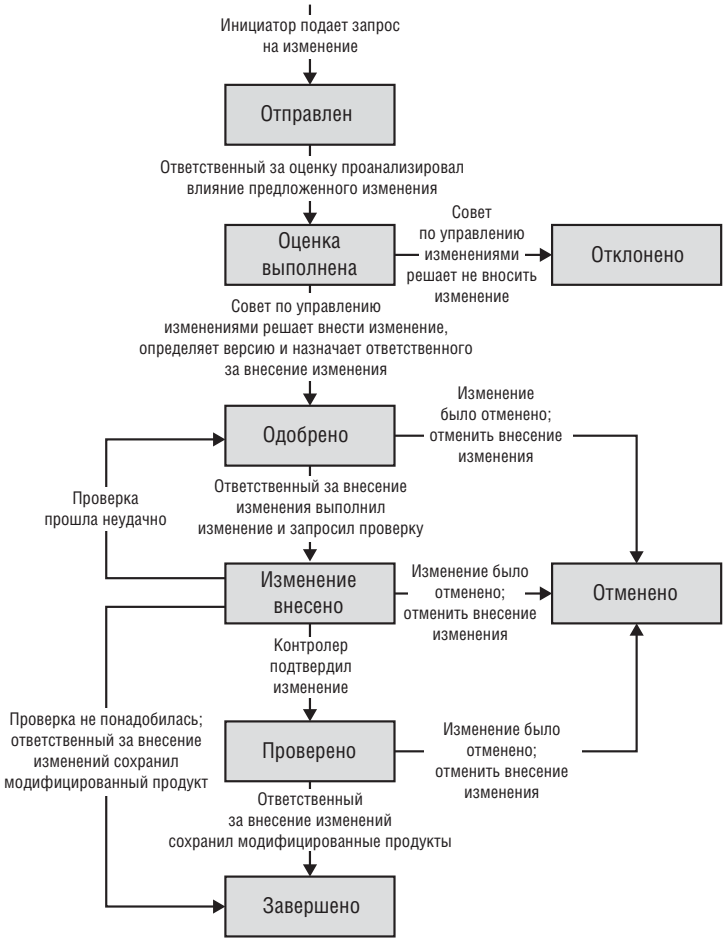


Рис. 28-2. Диаграмма перехода состояний для запроса на изменение

4. Входные критерии

Основным входным критерием для процесса контроля изменений является запрос на изменение, полученный по утвержденным каналам. Все потенциальные инициаторы запросов должны знать, как подавать запросы на изменение. Средство управления изменениями должно назначать каждому запросу на изменение уникальный идентификатор и направлять их всех к единой точке контакта — получателю запроса.

5. Задачи

В этом разделе процесса описываются задачи, выполняемые при обработке одного запроса на изменение.

5.1. Оценка запроса на изменение

Начните с оценки запроса на предмет технической выполнимости, затрат и соответствия бизнес-требованиям проекта и ограничениям ресурсов. Председатель совета по управлению изменениями может назначить ответственного за оценку влияния изменения, анализа риска, анализа опасности и др. (См. раздел «Анализ влияния изменений» далее в этой главе.) Анализ позволяет убедиться, что последствия изменения ясны. Ответственный за оценку, и члены совета по управлению изменениями должны также рассмотреть коммерческие и технические последствия отклонения изменения.

5.2 Принятие решения об изменении

Далее уполномоченные члены совета по управлению изменениями решают, утвердить или отклонить запрошенное изменение. Совет по управлению изменениями присваивают каждому одобренному изменению приоритет или назначает дату реализации или же назначает это изменение в определенную итерацию или выпуск. Совет может просто добавить новое требование в резерв продукта. После совет обновляет состояние запроса и информирует об этом всех заинтересованных членов команды.

5.3. Реализация изменения

После этого ответственный (или ответственные) за внесение изменений обновляет рабочие продукты с тем, чтобы реализовать изменение. Используйте информацию о связях требований, чтобы найти все части системы, затронутые изменениями, и при необходимости обновите информацию о связях в соответствии с этим.

5.4. Проверка изменения

Изменения требования, как правило, проверяют с помощью рецензирования, чтобы убедиться, что измененные требования, варианты использования и модели соответствующим образом отражают все аспекты изменения.

Несколькие члены команды могут проверить изменения с помощью тестирования или рецензирования. После этих процедур ответственный за внесение изменений сохраняет обновленные продукты в соответствующих местах, определенных проектными документами и соглашениями об управлении кодом.

6. Выходные критерии

Все перечисленные далее выходные критерии должны быть удовлетворены, чтобы считать процесс управления изменениями должным образом завершенным:

- состояние запроса: «Отклонено», «Закрыто» или «Отменено»;
- все измененные продукты обновлены и сохранены в соответствующих местах;
- все заинтересованные лица оповещены о деталях изменения и текущем состоянии запроса на изменение.

7. Отчет о состоянии контроля изменений

Определите графики и отчеты, которые вы будет использовать при обобщении содержимого базы данных контроля изменений. Обычно на этих графиках показано количество запросов на изменение в каждой категории состояния как функция времени или динамика среднего времени, когда запрос на изменение остается неразрешенным. Опишите процедуры создания графиков и отчетов. Менеджер проекта использует эти отчеты при контроле состояния проекта.

Приложение. Атрибуты запросов на изменение

В табл. 28-2 перечислены некоторые атрибуты данных, которые можно хранить для каждого запроса на изменение. Одну часть этих атрибутов задает инициатор, а вторую — совет по управлению изменениями. Укажите, какие атрибуты обязательные, а какие — нет. Не определяйте больше атрибутов, чем нужно. Частью этих атрибутов (идентификатор, даты создания и обновления) средство управления изменениями должно управлять автоматически.

Табл. 28-2. Предлагаемые атрибуты запросов на изменение

Атрибут	Описание
Происхождение изменения	Функциональная область, к которой относится изменение; это могут быть маркетологи, руководство, клиенты, разработчики или тестировщики
Идентификатор запроса на изменение	Уникальный идентификатор, назначенный запросу
Тип изменения	Тип запроса на изменение, например изменение требования, предложенное улучшение или отчет об ошибке

Табл. 28-2. (*окончание*)

Атрибут	Описание
Дата подачи	Дата, когда инициатор отправил запрос на изменение
Дата обновления	Дата последнего изменения запроса
Описание	Текстовое описание в свободной форме запрошенного изменения
Приоритет при реализации	Относительная важность внесения изменения, определенная советом по управлению изменениями — низка, средняя или высокая
Ответственный за внесение изменений	Лицо отвечающее за внесение изменения
Инициатор	Лицо, подавшее запрос на изменение
Приоритет инициатора запроса	Относительная важность внесения изменения с точки зрения инициатора: низка, средняя или высокая
Планируемый выпуск	Версия продукта или номер сборки, на которую запланировано одобренное изменение
Проект	Название проекта, для которого предлагается изменение
Ответ	Ответ в свободной текстовой форме на запрос об изменении; со временем их может быть несколько; не изменяйте существующие ответы при вводе нового
Состояние	Текущее состояние запроса на изменение, выбранное из возможных на рис. 28-2
Название	Краткое (одна строчка) описание предложенного изменение
Контролер	Лицо, определяющее, корректно ли реализовано изменение

Совет по управлению изменениями

Совет по управлению изменениями (change control board, CCB) — это группа людей, независимо от того, сколько их — один человек или несколько, принимающие решение о том, какие предложенные изменения требований и новые требования принять как есть или с корректировками, и от каких отказаться. Совет по управлению изменениями также решает, какие выявленные ошибки следует исправить и когда. Некоторые советы имеют полномочия для принятия решений, и они просто информируют менеджеров о внесении этих изменений, тогда как другие могут только давать рекомендации менеджерам для принятия решений. Многие проекты уже де-факто имеют такие группы; официальное назначение совета по управлению изменениями позволяет определить его структуру и полномочия, а также назначить рабочие процедуры.

Для некоторых людей термин «совет по управлению изменениями» означает неэкономичные бюрократические накладные расходы. Однако это ценная структура, помогающая управлять даже небольшим проектом. В небольшом проекте имеет смысл делегировать принятие решений одному или двум сотрудникам. В очень крупных проектах или программах может быть несколько уровней советов контроля изменений: одни отвечают за принятие бизнес-решений, например об изменении требований, а другие — за решения технического характера. При разработке большой программы, объединяющей несколько проектов, следует создать совет, принимающий решения на уровне программы, и отдельные советы для каждого проекта. Последние разрешают проблемы и рассматривают изменения, влияющие только на конкретный проект. Вопросы, касающиеся других проектов, и изменения, в результате которых возможно превышение затрат или нарушение графика, передаются в совет уровня программы.

Состав совета по управлению изменениями

Члены совета по управлению изменениями представляют все группы, которым необходимо принимать участие в принятии решений в рамках полномочий совета. Есть смысл привлечения к участию представителей следующих областей:

- менеджеры продукта или аналитики требований;
- бизнес-аналитики и менеджеры продукта;
- разработчики;
- тестировщики;
- маркетологи, компании, для которых создается приложение, или представители клиента;
- сотрудники отдела технической поддержки и сопровождения.

Только некоторые из них будут принимать решения, однако всех заинтересованных лиц следует проинформировать о решениях, влияющих на их работу. Если проект распространяется и на ПО, и на оборудование, в совет по управлению изменениями можно ввести специалистов по оборудованию, системных инженеров, специалистов по производству. Совет должен быть небольшим, чтобы можно было быстро и эффективно реагировать на запросы на изменение. Убедитесь, что члены совета понимают свои обязанности и относятся к ним серьезно. Для того чтобы убедиться, что совет владеет необходимой технической и бизнес-информацией, пригласите специалистов на совещание, где будут обсуждаться предложения, относящиеся к сфере деятельности этих специалистов.

Устав совета по управлению изменениями

Все проектные команды в организации должны следовать одному процессу управления изменениями, но советы в проектах могут действовать по-

разному. В каждом проекте нужно создать короткий устав (он может быть частью плана по управлению проектом), где описываются задачи, область полномочий, членство, производственные процедуры и процесс принятия решений совета по управлению изменениями (Sorensen, 1999). Шаблон устава совета можно загрузить с веб-сайта книги. В уставе должна быть указана регулярность запланированных совещаний и условия, инициирующие созыв специальных встреч. Область полномочий указывает, какие решения совет может принимать, а какие следует передать совету более высокого уровня.

Принятие решений

Каждый совет по управлению изменениями должен определить процесс принятия решения, в описании которого следует указать следующее:

- сколько членов совета по управлению изменениями или лиц, выполняющих ключевые роли, составляют кворум для принятия решений;
- правила принятия решений (подробнее о правилах принятия решений см. главу 2);
- может ли председатель совета по управлению изменениями отклонить коллективное решение совета;
- должен ли совет по управлению изменениями более высокого уровня или кто-либо еще ратифицировать решение.

Совет по управлению изменениями взвешивает предполагаемые преимущества и предполагаемое влияние принятия предложенного изменения. К преимуществам относятся: экономия средств, увеличение дохода, большее удовлетворение покупателей и конкурентные преимущества. К возможным негативным эффектам относятся: увеличение затрат на разработку и поддержку, задержка поставки и снижение качества продукта.

> **Внимание!** Поскольку люди не любят говорить «нет», то легко собрать огромный резерв одобренных запросов на изменение, которые никогда не будут реализованы. Прежде чем принять изменение, убедитесь, что вы понимаете его логическое обоснование и бизнес-ценность.

Информирование о состоянии

После того, как совет по управлению изменениями принимает решение, уполномоченный сотрудник обновляет состояние запроса в базе данных изменений. Некоторые средства автоматически генерируют сообщения электронной почты, чтобы информировать о новом состоянии инициатора запроса и всех, кого затрагивает это изменение. Если сообщение электронной почты не генерируется автоматически, проинформируйте заинтересованных специалистов лично для того, чтобы они смогли должны образом среагировать на это изменение.

Пересмотр обязательств

Было бы глупо предполагать, что заинтересованные лица могут включать все больше и больше функциональности в проект, который ограничен графиком, кадрами, бюджетом и качеством, и проект все равно останется успешным. До принятия важного изменения требования повторно обговорите обязательства с менеджерами и клиентами, чтобы принять это изменение. Вы можете просить увеличить сроки или отложить низкоприоритетные требования. Если вы не добьетесь определенного пересмотра обязательств, составьте список рисков проекта, ставящих под угрозу его успех, чтобы люди не удивлялись, если результаты не будут соответствовать желаемым.

Средства управления изменениями

Многие команды используют серийные средства отслеживания дефектов для сбора, хранения и управления изменениями требований. Список предложений изменений, поданных за последнее время, сгенерированный с помощью такого средства, может служить повесткой дня совещания совета по управлению изменениями. Средства отслеживания дефектов также можно использовать для отчета о количестве запросов в каждой категории состояния в любой момент времени. Так как перечень поставщиков, доступных на рынке средств и их функциональности часто меняется, мы не даем рекомендаций по выбору инструментов для управления изменениями. Обращайте внимание на следующие параметры средства для поддержки процесса изменения требований:

- позволяет определить атрибуты запроса на изменение;
- позволяет определить модель перехода состояния для жизненного цикла запроса на изменение;
- приводит в действие модель перехода состояния, чтобы только уполномоченные пользователи смогли вносить изменения в состояния;
- фиксирует дату изменения каждого состояния и лицо, выполнившее это изменение;
- позволяет рассылать автоматические уведомления по электронной почте при подаче нового или обновлении существующего запроса;
- создает стандартные и нестандартные отчеты и графики.

В некоторые коммерческие средства управления требованиями встроена простая система внесения изменения. Эти системы могут связать предложенное изменение с определенным требованием, чтобы сотрудник, ответственный за каждое требование, получал уведомление по электронной почте каждый раз при поступлении соответствующего запроса на изменение.

Оснащение процесса

Когда я работал в команде, занимающейся веб-разработками, одним из наших первых улучшений стала реализация контроля изменений для управления большим резервом запросов на изменение (Wiegers, 1999). Мы начали с процесса, похожего на один из тех, что описан в этой главе. Мы пробовали его в течение нескольких недель, используя бумажные формы, пока я оценивал несколько программных средств отслеживания дефектов. Мы нашли несколько способов улучшения нашего процесса и обнаружили несколько дополнительных элементов данных, которые требовались нам в запросах на изменение. Мы выбрали отлично конфигурируемое средство отслеживания дефектов и настроили его с учетом специфики нашего процесса. Команда использовала этот процесс и средство для обработки изменений требований в системе, касающихся разработки, отчетов об ошибках и предложенных улучшений в производственных системах, обновлений содержимого веб-сайта и запросов, касающихся новых проектов по разработке. Управление изменениями стало одной из наших наиболее успешных инициатив по улучшению процесса.

Измерение активности изменений

Измерение активности изменений — это способ оценки стабильности требований. Он позволяет также обнаружить возможности для улучшения процесса, чтобы снизить количество изменений в будущем. Возможно, вам следует контролировать следующие показатели активности изменений:

- общее количество полученных запросов на изменение, открытых и закрытых в настоящее время;
- совокупное число пришедших запросов, включая добавленные, удаленные и измененные требования;
- количество полученных запросов на изменение, исходящих из каждого источника;
- количество изменений, предложенных и внесенных в каждое требование после создания базовой версии;
- общие усилия на обработку и реализацию запросов на изменение.

Необязательно настолько тщательно отслеживать этапы изменения требований. Как и в случае с показателями ПО, необходимо до принятия решения о том, что именно вы будете измерять, четко понять цель измерения и то, как вы будете использовать полученные данные (Wiegers, 2007). Начните с простых метрик, чтобы выработать культуру измерения в вашей организации и собрать ключевые данные, необходимые для эффективного управления проектами.

На рис. 28-3 показан один из способов отслеживания количества изменений требований в ходе разработки проекта (Wiegers, 2006). Этот график

изменчивости требований позволяет отследить скорость поступления новых предложений по изменению после определения базовой версии. Кривая на этом графике должна стремиться к нулю по мере приближения даты поставки. Высокая частота в течение длительного времени свидетельствует о том, что возможно нарушение расписания поставки продукта. Вероятна и другая причина: первоначальная базовая версия требований была неполной, то есть следует улучшить подход к сбору информации по требованиям.

Рис. 28-3. Пример графика активности изменения требований

Также полезна информация об отслеживании источников изменения требований. На рис. 28-4 показан способ представления количества запросов на изменение, пришедших из разных источников. Менеджер проекта может обсудить такой график с менеджером по маркетингу, так как большинство изменений запрошено именно отделом маркетинга. Плодотворная дискуссия позволит установить, какие действия должна предпринять команда, чтобы уменьшить число изменений, полученных после создания базовой версии из отдела маркетинга. Использование данных в качестве отправной точки для такой дискуссии гораздо продуктивнее встречи противоборствующих сторон, основанной на взаимном раздражении. Предложите свой список возможных источников изменения требований.

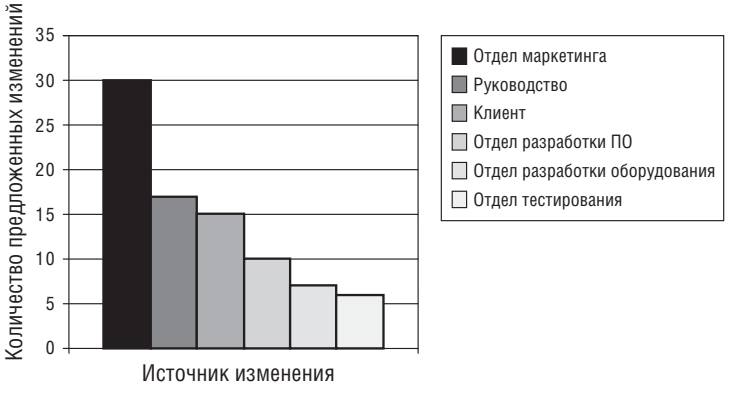


Рис. 28-4. Пример графика, демонстрирующего источники изменения требований

Анализ влияния изменений

Необходимость анализа результатов изменения при внесении значительных изменений очевидна. Однако неожиданные сложности могут вызывать даже самые незначительные запросы на изменение. Как-то одной компании понадобилось изменить в продукте текст одного сообщения об ошибке. Казалось, что может быть проще? Существовали две версии продукта — английская и немецкая. В английской версии все было в порядке, но в немецкой размер сообщения превышал максимально допустимую длину сообщения об ошибке и в окне сообщения и в базе данных. Выполнение этого на первый взгляд простого задания оказалось более трудоемким, чем предполагал разработчик, когда обещал быстрое решение.

Анализ влияния — это ключевой аспект ответственного управления требованиями (Arnold и Bohner, 1996). Он обеспечивает точное понимание подтекста предложенного изменения, что помогает команде принимать информированные бизнес-решения о том, какое изменение одобрить. Анализ позволяет выявить компоненты, которые может понадобиться создать, изменить или отклонить, и оценить затраты, связанные с реализацией изменения. До того, как разработчик ответит: «Конечно, без проблем», он должен потратить время на анализ результата изменения.

Процедура анализа влияния изменения

Председатель совета по управлению изменениями обычно просит одно или нескольких технически специалистов (бизнес-аналитика, разработчиков и/или тестировщиков) выполнить анализ влияния определенного изменения. Анализ результатов изменений предусматривает три шага:

1. Понимание возможных последствий изменения. Часто они распространяются, как круги на воде, вызывая изменения в других требованиях, архитектурах, дизайне, коде и тестах. Изменения могут вести к конфликтам с другими требованиями или ухудшать атрибуты качества, такие как производительность и безопасность.
2. Определите все требования, файлы, модели и документы, которые, возможно, придется изменить, если команда включит все запрошенные изменения.
3. Определите задачи, необходимые для реализации изменения, и оцените усилия, необходимые для выполнения этих задач.

> **Внимание!** Пропустив выполнение анализа влияния, вы не измените объем задачи. Просто в этом случае этот объем станет сюрпризом для вас. А сюрпризы в области ПО редко бывают приятными.

На рис. 28-5 показан контрольный список вопросов, разработанный в помощь ответственному за оценку последствий принятия предлагаемого изменения. В списке на рис. 28-6 перечислены наводящие вопросы, которые помогут вам определить все элементы ПО и другие результаты работы, на кото-

рые может повлиять изменение. Информация о связях требований с другими основными документами проекта, крайне полезны при анализе влияния. По мере наработки опыта при использовании этих контрольных вопросов адаптируйте их для ваших проектов. (Рисунки с 28-5 по 28-8 можно загрузить с веб-сайта книги *http://aka.ms/SoftwareReq3E/files*)

- ❑ Улучшит или ухудшит ли изменение способность удовлетворять каким-либо бизнес-требованиям?
- ❑ Конфликтуют ли какие-то из требований в базовой версии с предложенным изменением?
- ❑ Конфликтуют ли какие-то из отложенных требований в базовой версии с предложенным изменением?
- ❑ Каковы технические или бизнес-последствия невнесения изменения?
- ❑ Какие могут быть побочные негативные эффекты или другие риски, если изменение не будет внесено?
- ❑ Повлияет ли негативно предложенное изменение на требования к производительности и другие атрибуты качества?
- ❑ Выполнимо ли предложенное изменение в рамках известных технических ограничений и квалификации специалистов?
- ❑ Не потребуется ли для реализации предложенного изменения неприемлемое количество каких-либо ресурсов, необходимых для разработки, тестирования или рабочих сред?
- ❑ Нужно ли приобрести какие-либо средства для реализации и тестирования этого изменения?
- ❑ Как предложенное изменение повлияет на последовательность, зависимости, усилия или продолжительность задач, включенных в настоящее время в план проекта?
- ❑ Потребуется ли создание прототипов или другая информация пользователей для проверки предложенного изменения?
- ❑ Сколько затрат, уже вложенных в проект, будут потеряны, если принять это изменение?
- ❑ Не увеличится ли затрата на единицу продукта при принятии изменения, например за счет увеличения расходов на покупку лицензий на продукты сторонних разработчиков?
- ❑ Повлияет ли изменение на планы по маркетингу, производству, обучению или поддержки клиентов?

Рис. 28-5. Вопросы, позволяющие понять возможные последствия предлагаемого изменения

- ❑ Определите все необходимые изменения пользовательского интерфейса, дополнения или удаления.
- ❑ Определите все изменения, добавления или удаления, которые необходимо внести в отчеты, базы данных или файлы.
- ❑ Определите компоненты дизайна, которые придется создать, изменить или удалить.
- ❑ Определите файлы исходного кода, которые необходимо создать, изменить или удалить.
- ❑ Определите все изменения, которые придется внести в файлы или процедуры построения.
- ❑ Определите существующие тесты модулей, интеграции и системные тесты, которые потребуется изменить или удалить.
- ❑ Оцените необходимое количество новых тестов модулей, интеграции и системных тестов.
- ❑ Определите все экраны справочной системы, обучающие материалы или другую пользовательскую документацию, которую необходимо создать или изменить.
- ❑ Определите все компоненты приложений, библиотек или оборудования, на которые повлияет изменение.
- ❑ Определите все стороннее ПО, которое потребуется приобрести или лицензировать.
- ❑ Определите влияние, которое предложенное изменение окажет на планы управления проектом ПО, проверки приемлемости качества, управления конфигурацией и другие планы.

Рис. 28-6. Контрольный список определения рабочих продуктов, на которые может повлиять предлагаемое изменение

Часы	Задача
_____	Обновить спецификацию требований или базу данных требований.
_____	Разработать и оценить прототип.
_____	Создать новые компоненты дизайна.
_____	Изменить существующие компоненты дизайна.
_____	Разработать новые компоненты пользовательского интерфейса.
_____	Изменить существующие компоненты пользовательского интерфейса.
_____	Разработать новую пользовательскую документацию и экраны справки.
_____	Разработать новую пользовательскую документацию и формы справочной системы.
_____	Разработать новый исходный код.
_____	Изменить существующий исходный код.
_____	Приобрести и встроить стороннее ПО.
_____	Изменить файлы и процедуры сборки.
_____	Разработать новые модульные тесты и тесты интеграции.
_____	Изменить существующие модульные тесты и тесты интеграции.
_____	Выполнить модульные тесты и тесты интеграции после реализации.
_____	Написать новые системные и приемочные тесты.
_____	Изменить новые системные и приемочные тесты.
_____	Изменить автоматизированные средства тестирования.
_____	Выполнить регрессивное тестирование.
_____	Разработать новые отчеты.
_____	Изменить существующие отчеты.
_____	Разработать новые элементы базы данных.
_____	Изменить существующие элементы базы данных.
_____	Разработать новые файлы данных.
_____	Изменить существующие файлы данных.
_____	Изменить различные планы проекта.
_____	Обновить остальную документацию.
_____	Обновить матрицу связей требований.
_____	Просмотреть измененные продукты.
_____	Переработать продукт после рецензирования и тестирования.
_____	Другие задачи.
_____	Всего затрат

Рис. 28-7. Таблица расчета затрат на изменение требования

Множество проблем с оценкой возникает из-за того, что специалист, выполняющий оценку, не принимает во внимание всю работу, необходимую для завершения задачи. Следовательно, этот подход к анализу влияния требует тщательного определения задачи. Для оценки значительных изменений используйте небольшие команды — а не просто одного разработчика: так вам удастся не упустить из внимания важные задачи. Далее описана несложная процедура оценки влияния предложенного изменения требования.

1. Пройдите контрольный список на рис. 28-5.
2. Пройдите контрольный список на рис. 28-6. В некоторые средства управления требованиями входят отчеты об анализе влияния, в которых размещены ссылки связей требований; следуя по ним удается найти элементы системы, зависящие от изменяемых требований.

3. Воспользуйтесь рабочей таблицей на рис. 28-7 для расчета затрат, необходимых для выполнения предполагаемых задач. Для выполнения большинства запросов на изменение потребуется только часть задач из рабочей таблицы.
4. Просуммируйте все затраты.
5. Определите последовательность выполнения задач и то, как они могут пересекаться с уже запланированными задачами.
6. Оцените влияние предложенного изменения на график и затраты.
7. Оцените приоритет изменения по сравнению с остальными требованиями.
8. Доложите о результатах анализа влияния совету по управлению изменениями.

В большинстве случаев эта процедура не должна отнимать более пары часов на один запрос на изменение. Для занятого разработчика это может показаться и много, однако это небольшая плата за уверенность в том, что ограниченные ресурсы используются обдуманно. Чтобы лучше оценивать влияние будущих изменений, сравните действительные затраты, необходимые для реализации каждого изменения, с оценкой затрат. Выясните причины всех различий и измените соответствующим образом списки оценки вопросов и рабочую таблицу, чтобы в дальнейшем точнее анализировать влияние.

Деньги на ветер

Два разработчика в компании A. Datum Corporation рассчитали, что уйдет четыре недели на добавление улучшения к одной из их информационных систем. Клиент утвердил их расчеты, и разработчики приступили к работе. Через два месяца работа была выполнена только наполовину, и клиент потерял терпение: «Если бы я знал, сколько времени это займет и сколько это будет стоить, я бы не согласился на это. Не нужно ничего делать». Торопясь получить одобрение и приступить к работе, разработчики не выполнили надежных расчетов анализа влияния, которые позволили бы клиенту принять должное бизнес-решение. В результате компания потратила сотни часов работы, которой можно было бы избежать, если бы они не поленились выделить всего несколько часов на предварительный анализ.

Шаблон отчета об анализе влияния изменения

На рис. 28-8 показан шаблон отчета о результатах анализа влияния каждого изменения требования. Специалистам, реализующим изменение, понадобятся детали анализа и рабочая таблица планирования затрат, но совету требуется только итог результатов анализа. Как и любой другой шаблон, попробуйте его в работе, а затем измените в соответствии со спецификой вашего проекта.

Идентификатор запроса на изменение _____
Название: _____
Описание: _____
_____
Ответственный за оценку: _____
Дата создания: _____
Общая оценка затрат: _____ рабочих часов
Оценка влияния на график: _____ дней
Дополнительные затраты: _____ долларов
Влияние на качество: _____
_____
Другие компоненты, затрагиваемые изменением: _____
_____
Другие задачи, затрагиваемые изменением: _____
Проблемы стоимости жизненного цикла: _____

Рис. 28-8. Шаблон отчета об анализе влияния

Управление изменениями в проектах гибкой разработки

Сама структура проектов гибкой разработки предусматривает и даже приветствует изменения границ проекта. Один из 12 принципов гибкой разработки ПО гласит: «Принимайте меняющиеся требования, даже на поздних стадиях разработки. Процессы ставят изменения на службу повышения конкурентных преимуществ клиента»» (*www.agilemanifesto.org/principles.html*). Этот принцип признает ту реальность, что изменения требований неизбежны, необходимы и часто ценны. Принятие изменений помогает достигать меняющиеся бизнес-цели и учитывать ограничения человеческого планирования и прогнозирования человеком.

В проектах гибкой разработки изменениями управляют, поддерживая динамический резерв (backlog) оставшейся работы (см. рис. 28-9). Под «работой» подразумеваются еще не реализованные пользовательские истории, не устраненные дефекты, не внесенные в бизнес-процессы изменения, не разработанные курсы обучения и масса других действий, присутствующих в любом проекте разработки ПО. В каждой итерации реализуется ряд самых приоритетных задач из резерва. Когда заинтересованные лица запрашивают выполнение дополнительной работы, эти задачи поступают в резерв и приоритизируются вместе с уже имеющимися задачами. Приоритеты задач, еще не назначенных на конкретные итерации, могут меняться, и эти задачи в любой момент могут удаляться из резерва. Новая высокоприоритетная пользовательская история, назначенная на следующую итерацию, может вытеснить менее приоритетную такого же размера, ранее назначенную на эту итерацию. Тщательное управление объемом каждой итерации гарантирует ее выполнение вовремя и с высоким качеством.

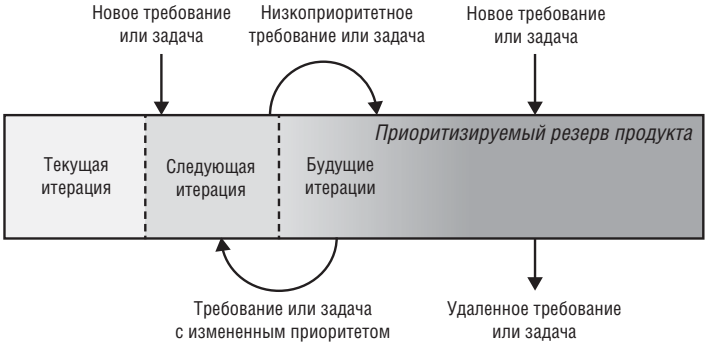


Рис. 28-9. В проектах гибкой разработки управление изменениями выполняется в динамическом резерве продукта

Ввиду итеративной природы проектов гибкой разработки, каждые несколько недель появляется возможность выбрать набор рабочих задач из резерва для следующей итерации разработки. В командах гибкой разработки по-разному работают с новыми задачами, появляющимися в процессе итерации: их могут откладывать на следующую итерацию или же менять состав текущей итерации. «Замораживание» состава задач на время выполнения итерации обеспечивает стабильность для разработчиков и предсказуемость для заинтересованных лиц. С другой стороны, корректировка содержимого итерации позволяет команде гибче реагировать на запросы клиента.

В этом отношении «философия» проектов гибкой разработки различается, но нет единого «правильного» подхода. Выберите, что лучше подойдет вашей команде и бизнес-целям проекта: фиксирование базовой версии на все время итерации или введение высокоприоритетных изменений при их выявлении. Основной принцип — избежать излишних изменений (модификация требований) или избыточной жесткости (фиксированные требования) в итерации. Одно решение заключается в определении правильной длительности итерации, чтобы минимизировать масштаб изменений в текущей итерации. Иначе говоря, если изменения приходится вносить слишком часто, может иметь смысл сократить стандартную длительность итерации.

Во всех методах гибкой разработки есть роль, представляющая конечного пользователя или клиента. В Scrum это роль владельца продукта, а в Extreme Programming — роль клиента. Клиент или владелец продукта — основное лицо, отвечающее за определение содержимого резерва продукта. Он также принимает решения о принятии предложенных изменений требований, основываясь на их соответствии общей концепции продукта и бизнес-ценности (Cohn, 2010).

Так как команда гибкой разработки является межфункциональной группой разработчиков, тестировщиков, бизнес-аналитиков, менеджера проекта и других, она уже обладает структурой описанного ранее в этой главе совета по управлению изменениями Короткая продолжительность итераций гибкой разработки и поставка продукта небольшими порциями позволяет ко-

мандам гибкой разработки выполнять управление изменениями часто, но в ограниченном масштабе. Однако даже в проектах гибкой разработки нужно оценивать возможные затраты, связанные с изменениями в требованиях, и влияние изменений на компоненты продукта. Об изменении границ проекта, способном повлиять на общие затраты и продолжительность проекта, нужно сообщать на более высокий уровень управления, например куратору проекта (Thomas, 2008).

Изменения будут происходить независимо от типа проекта или модели разработки, используемой в вашей команде. Нужно рассчитывать на них и быть готовым управлять ими. Четкое следование правилам управления изменениями может снизить масштаб нарушений, вносимых изменениями. Задача управления изменениями не в их подавлении и не в запрете заинтересованным лицам предлагать изменения, а в наглядности деятельности и механизмов, в рамках которых соответствующие люди рассматривают предлагаемые изменения и добавляют уместные изменения в проект в подходящее время. Это позволяет максимизировать бизнес-ценность и минимизировать негативное влияние изменений на работу команды.

Что дальше?

- Определите, кто будет принимать решения в вашем проекте, и назначьте их в совет по управлению изменениями. Примите устав, чтобы установить и задокументировать задачи совета, его состав и процесс принятия решений.
- Составьте диаграмму перехода состояний для жизненного цикла предложенных изменений требований на вашем проекте, начав с диаграммы на рис. 28-2. Опишите, как ваша команда будет поступать с предложенными изменениями требований. Вначале делайте это вручную, пока не убедитесь, что процесс практичен и эффективен.
- Выберите серийное средство отслеживания проблем или вопросов, совместимое с вашей операционной средой разработки. Измените его в соответствии с процессом контроля изменений, созданным на предыдущем этапе.
- В следующий раз, когда вы будете оценивать запрос на изменение требований, сначала рассчитайте затраты, используя старый способ. Затем, оцените его еще раз, используя прием анализа влияния, описанный в этой главе. Если вы реализуете изменение, сравните результаты обоих расчетов, чтобы понять, какие расчеты точнее показывают непосредственные затраты. Измените списки вопросов для анализа влияния и рабочую таблицу согласно полученному опыту.

Глава 29

Связи в цепи требований

«Мы только что обнаружили, что в новом контракте с профсоюзом изменены правила расчета оплаты сверхурочных часов и надбавок за работу в ночную смену, — сообщил Джастин на еженедельном совещании. — Они также изменили то, как трудовой стаж влияет на график отпусков, на льготы и все остальное. Нам нужно срочно обновить системы платежных ведомостей и графиков работы с учетом этих изменений. Крис, как ты думаешь, сколько времени это займет?»

«Ну, это большой объем работ, — сказал Крис. — Правила трудового стажа учитываются во всей системе составления графика. Я не могу назвать точный срок. У меня уйдут часы только на просмотр кода и попытку найти все фрагменты, где содержатся правила».

Кажущиеся простыми изменения ПО часто влияют на многие элементы продукта, поэтому возникает необходимость в их изменении. Трудно обнаружить все компоненты системы, которые могут быть затронуты модификацией требования. В главе 28 обсуждается важность анализа влияния — он позволяет убедиться, что команда понимает последствия изменения до принятия на себя обязательств по его выполнению. Выполнять анализ влияния проще, если вы построите «дорожную карту», на которой показано, где именно в ПО реализовано каждое требование и бизнес-правило.

В этой главе обсуждается отслеживание требований. Связи требований документируют зависимости и логические связи отдельных требований и других элементов системы. В эти элементы входят другие требования различных типов, бизнес-правила, архитектура и другие компоненты дизайна, модули исходного кода, тесты и файлы справки. Информация о связях облегчает выполнение анализа влияния, помогая определить все рабочие продукты, которые вам может понадобиться изменить для реализации предложенного изменения требования.

Отслеживание связей требований

Связи помогают следить за развитием требования в обоих направлениях — от первоисточника к реализации и наоборот. В главе 11 отслеживаемость

определяется, как одна из характеристик отличной спецификации требований. (Имейте в виду, что *отслеживаемость*, то есть наличие ссылок, облегчающих отслеживание, не одно и то же, что *наличие следов* — логических ссылок между требованиями и другими зафиксированными элементами.) Чтобы реализовать отслеживаемость, каждое требование должно быть уникально и последовательно идентифицировано, чтобы можно было ссылаться на него однозначно в ходе работы над проектом. Пишите требования маленькими фрагментами, а не большими абзацами, в которых содержится множество отдельных функциональных требований — это приводит к разбросу элементов дизайна и кода.

На рис. 29-1 показаны четыре типа связей отслеживания (Jarke, 1998). Потребности клиента отслеживаются *в прямом направлении к требованиям*, чтобы можно было определить, какие требования будут затронуты, если в течение или после разработки потребности изменятся. Потребности клиента могут выражаться в форме бизнес-целей, потребностей рынка и/или пользовательских требований. Полный набор прямых связей также дает уверенность, что в спецификации требований указаны все потребности клиента. И, наоборот, вы можете проследить *в обратном направлении от требований* к потребностям клиента, чтобы определить происхождение каждого требования к ПО. Если вы представите потребности клиента в форме вариантов использования, то верхняя половина рис. 29-1 будет показывать связи между вариантами использования и функциональными требованиями.

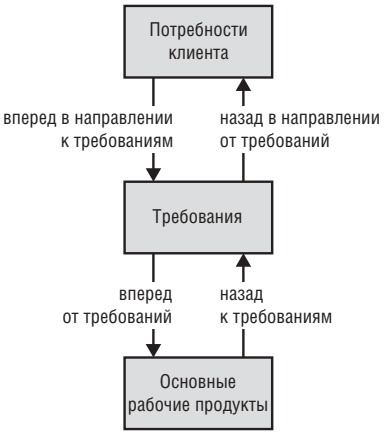


Рис. 29-1. Четыре типа отслеживаемости требований

В нижней части рис. 29-1 видно, что по мере приближения к поставке, вы можете отслеживать *вперед в направлении от требований*, определив связи между отдельными функциональными и нефункциональными требованиями и элементами продукта. Этот тип связи гарантирует, что каждое требование удовлетворено, поскольку вы знаете, какой компонент соответствует каждому требованию. Четвертый тип связи отслеживает отдельные элементы продукта *назад в направлении к требованиям*, чтобы вы знали причину

создания каждого элемента. В большинство приложений включен вспомогательный или поддерживающий код, не относящийся напрямую к требованиям пользователей, например для тестирования, но вы должны знать, почему написана каждая строка кода.

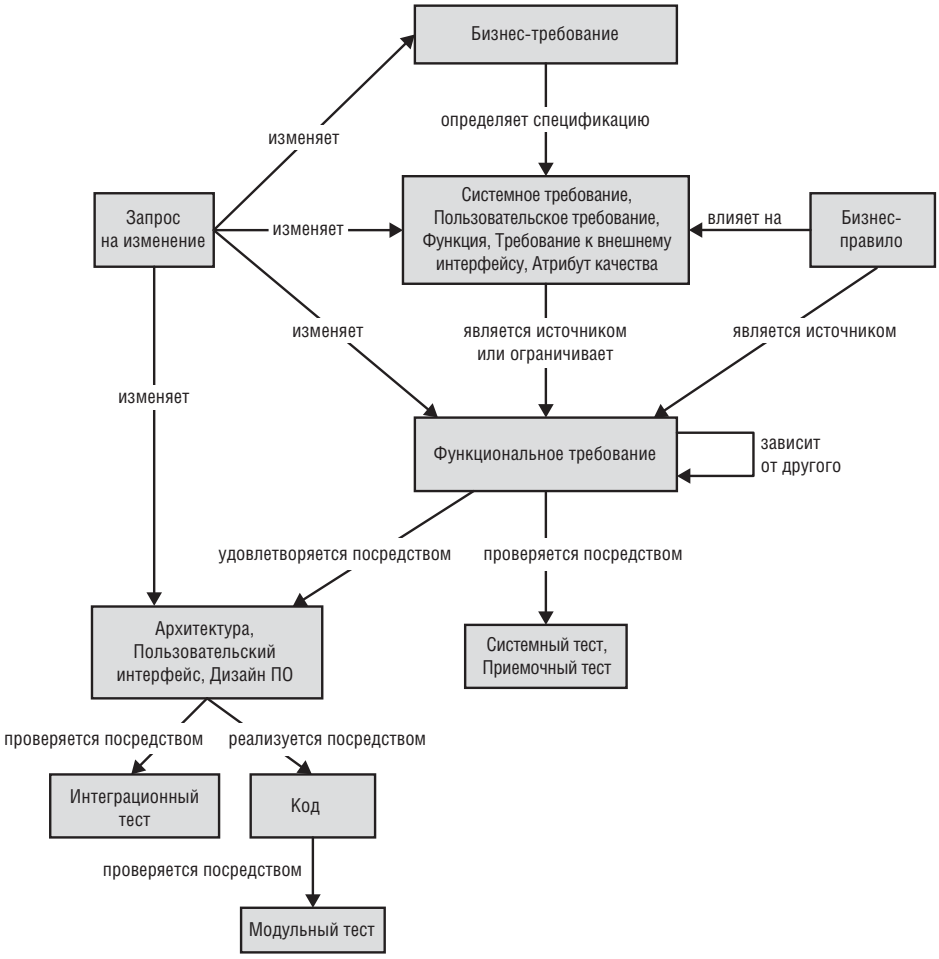


Рис. 29-2. Возможные связи отслеживания требований

Предположим, тестировщик обнаружил незапланированную функциональность при отсутствии соответствующего требования. Этот фрагмент кода может свидетельствовать, что разработчик реализовал официальное подразумеваемое или сформулированное устно требование, которое бизнес-аналитик теперь может добавить в спецификацию. Или же это может быть «висящий» код, украшающий фрагмент, который не относится к продукту. Связи отслеживаемости позволяют разобраться в подобных ситуациях и получить более полное представление о том, как именно фрагменты вашей системы составляют одно целое. И наоборот: тесты, которые созданы и отслеживаются до отдельных требований, также представляют собой меха-

низм выявления нереализованных требований, поскольку ожидаемой функциональности не будет в тестируемой системе. Связи отслеживаемости помогают отслеживать родительские требования, взаимосвязи и зависимости между отдельными требованиями. Эта информация показывает влияние изменения, если какое-то требование удаляется или модифицируется.

На рис. 29-2 показано множество типов прямых взаимосвязей отслеживаемости, возможных в проекте. Естественно, вам не нужно определять и управлять всеми этими типами связей. Во многих проектах удается получить большинство преимуществ отслеживаемости, приложив совсем немного усилий. Возможно, вам нужно только отслеживать тестирование системы до функциональных или пользовательских требований. Взвесив затраты и преимущества, решите, какие связи уместны в вашем проекте, и вы в значительной степени поспособствуете успешной разработке и эффективному обслуживанию. Не просите членов команды тратить много времени на фиксирование информации, если у вас нет четкого представления об ее использовании.

Мотивация для отслеживаемости требований

Однажды у меня был неудачный опыт: я написал программу, а потом понял, что случайно упустил из виду требование. Оно было в спецификации — я его просто не заметил. Мне пришлось вернуться и написать дополнительный код, а я думал, что уже закончил с программированием. Пропуск требования — это нечто большее, чем просто промах, если клиент не удовлетворен или в готовом продукте отсутствует критически важная функция. Отслеживание требований служит способом продемонстрировать соответствие контракту, спецификации или правилу. Это позволяет улучшить качество продуктов, снизить затраты на поддержку и облегчить повторное использование.

Для обновления информации о связях по мере разработки и обслуживания системы необходима дисциплина. Если информация отслеживаемости устаревает, скорее всего ее не удастся заново воссоздать. Устаревшая или неточная информация о связях заставляет разработчиков и сотрудников службы поддержки тратить время впустую, направляя их по неверному пути. Именно поэтому для использования отслеживаемости у вас должна быть веская причина (Ramesh и др., 1995). Далее перечислены возможные преимущества реализации отслеживаемости требований.

- **Обнаружение пропущенных требований** Поищите бизнес-требования, которые не связаны ни с какими пользовательскими требованиям, и пользовательские требования, которые не связаны с какими-либо функциональными требованиями.

- **Поиск ненужных требований** Поищите функциональные требования, не связанные с пользовательскими или бизнес-требованиями и, поэтому, ненужные.
- **Сертификация и выполнение нормативов** Вы можете воспользоваться информацией отслеживаемости при сертификации продукта с особыми требованиями к безопасности, чтобы продемонстрировать, что все требования были реализованы — хотя это не доказывает, что они реализованы корректно и полностью! Информация отслеживания показывает, что требования, необходимые для удовлетворения нормативных требований включены и выполнены, что обычно требуется для компаний, работающих в области здравоохранения и финансов.
- **Анализ влияния изменения** Без информации о связях возрастает вероятность, что из внимания будет упущен системный элемент, которого коснется добавление, удаление или изменение определенного требования.
- **Поддержка и обслуживание** Надежная информация отслеживаемости облегчает соответствующее и полное внесение изменений в ходе обслуживания. Часто при изменении корпоративных политик или законодательства, приложения нуждаются в обновлении. Таблица, в которой показано, где соответствующее бизнес-правило было реализовано в функциональном требовании, дизайне и коде, упрощает соответствующее внесение изменений.
- **Отслеживание проекта** Если в ходе разработки вы тщательно фиксируете данные отслеживаемости, у вас будет точное представление о состоянии реализации запланированной функциональности. Отсутствующие связи указывают на рабочие продукты, которые еще не созданы.
- **Реинжиниринг** Вы можете перечислить функции в предыдущей версии системы, которую вы заменяете, и зафиксировать, с какими новыми системными требованиям и компонентами ПО они связаны.
- **Повторное использование** Информация отслеживаемости упрощает повторное использование компонентов продукта, определяя пакеты связанных требований, дизайна, кода и тестов.
- **Тестирование** Если тестирование дает отрицательный результат, то связи между тестами, требованиями и кодом укажут на наиболее вероятные части кода, которые необходимо проверить на наличие дефектов.

Многие из вышеперечисленных относятся к долгосрочным выгодам, которые снижают общую стоимость жизненного цикла продукта, хотя при этом увеличиваются затраты на сбор и управление информацией отслеживаемости. Рассматривайте отслеживаемость требований как инвестицию, увеличивающую шансы поставить продукт, который удовлетворяет все основные требования клиента. Вы будете получать дивиденды каждый раз, когда вам понадобится изменить, расширить или заменить продукт. Определить связи отслеживаемости несложно, если вы собираете информацию по мере разработки, однако делать это для уже завершенной системы утомительно и дорого.

Матрица отслеживаемости требований

Наиболее типичный способ представления связей между требованиями и другими элементами системы — *матрица отслеживаемости требований*, которую также называют таблицей трассируемости (requirements traceability matrix). Джой Битти и Энтони Чен описывают аналогичное средство, которое называется *матрицей сопоставления требований* (requirements mapping matrix) и показывает связи между требованиями и различными другими объектами. В табл. 29-1 показана часть такой матрицы для Chemical Tracking System. Когда я раньше создавал такие матрицы, я делал копию базовой версии спецификации требований и удалял все, кроме идентификаторов функциональных требований. Затем я создавал таблицу, отформатированную так же, как табл. 29-1, и заполнял только столбец «Функциональное требование». Далее мы постепенно заполняли пустые ячейки в матрице по мере разработки проекта.

Табл. 29-1. Один из типов матрицы отслеживаемости требований

Пользовательское требование	Функциональное требование	Элемент дизайна	Элемент кода	Тест
UC-28	catalog.query.sort	Каталог классов	CatalogSort()	search.7
				search.8
UC-29	catalog.query.import	Каталог классов	CatalogImport()	search.12
			CatalogValidate()	search.13
				search.14

В табл. 29-1 показано, как каждое функциональное требование связано с определенным вариантом использования (в направлении назад) и с одним или более элементами дизайна, кода и тестирования (в направлении вперед). Элементы дизайна могут быть объектами в таких моделях анализа, как диаграммы потока данных, таблицы в реляционной модели данных или классах объектов. Ссылки кода могут быть методами класса, хранимыми процедурами, именами файлов исходного кода, а также модулями в исходном файле. Вы вправе добавить дополнительные столбцы для расширения ссылок на другие рабочие продукты, например на документацию справочной системы. Добавление деталей отслеживаемости — дополнительная работа, но так вы получаете точные расположения связанных элементов ПО.

Заполняйте информацию по мере выполнения работы, а не по мере планирования. То есть вводите CatalogSort() в столбец «Элемент кода» первой строки в табл. 29-1, только когда код в этой функции написан. Таким образом, читающий матрицу будет знать, что заполненные ячейки матрицы для отслеживания требований указывают на работу, которая уже выполнена.

> **Внимание!** Перечисление вариантов тестирования для каждого требования *не* указывает на то, что ПО протестировано. Это просто означает, что в соответствующее время были написаны определенные тесты для проверки требований. Отслеживание состояния тестов — это отдельный вопрос.

Другой способ представить информацию отслеживаемости — с помощью набора матриц, определяющих связи между парами элементов системы, например:

- один тип требования с другим требованием этого же типа;
- один тип требования с требованием другого типа;
- один тип требования с тестами.

Вы можете использовать эти матрицы для определения различных взаимоотношений, возможными между парами требований, например «указывает/указан посредством», «зависит от», «является родительским для» и «ограничивает/ограничен посредством» (Sommerville и Sawyer, 1997).

В табл. 29-2 показана двусторонняя матрица отслеживаемости. Большинство ячеек матрицы не заполнены. Каждая ячейка на пересечении двух связанных компонентов указывает на наличие соединения. В табл. 29-2 стрелка указывает, что данное функциональное требование отслеживается от определенного варианта использования. Например, у требования FR-2 есть связь от UC-1, а у требования FR-5 — от UC-2 и UC-4. Это говорит, что функциональное требование FR-5 повторно используется в двух вариантах использования — UC-2 и UC-4.

Табл. 29-2. Матрица для отслеживания требований, показывающая связи между вариантами использования и функциональными требованиями

	Вариант использования			
Функциональное требование	**UC-1**	**UC-2**	**UC-3**	**UC-4**
FR-1	↵			
FR-2	↵			
FR-3			↵	
FR-4			↵	
FR-5		↵		↵
FR-6			↵	

Связи отслеживаемости могут определить отношения «один к одному», «один ко многим» или «многие ко многим» между элементами системы. Формат в табл. 29-1 предусматривает это, позволяя вводить несколько позиций в каждой ячейке таблицы. Ниже приведены примеры возможных связей.

- **«Один к одному»**: один элемент дизайна реализуется в одном модуле кода;
- **«Один ко многим»**: одно функциональное требование проверяется множеством вариантов тестирования;
- **«Многие ко многим»**: каждый вариант использования порождает множество функциональных требований, а определенные функциональные требования являются общими для нескольких вариантов использования. Подобным образом общие или повторяющиеся элементы дизайна могут удовлетворять нескольким функциональным требованиям. В идеале стоило бы зафиксировать все эти взаимосвязи, однако на практике отношениями отслеживаемости типа «многие ко многим» сложно и трудно управлять.

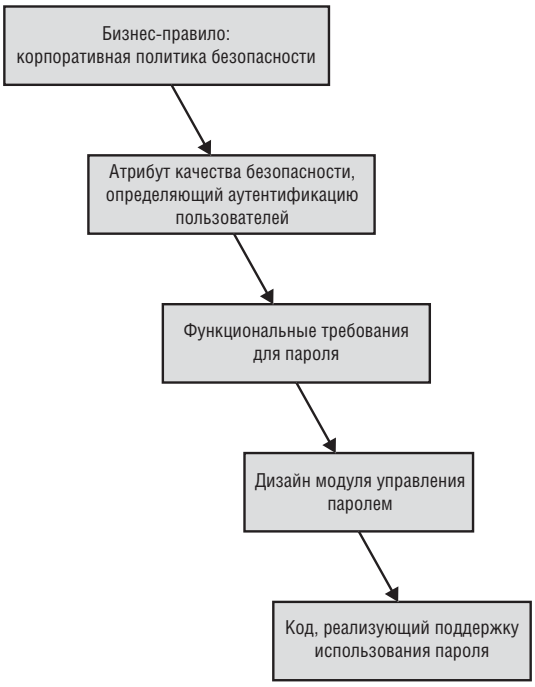


Рис. 29-3. Пример цепи отслеживаемости для требований, касающихся безопасности приложения

Нефункциональные требования, такие как атрибуты качества не всегда прослеживаются напрямую до кода. Требование к времени отклика может диктовать выбор определенного оборудования, алгоритмов, структур баз данных или архитектуры. Требование к легкости перемещения может ограничить функции языка, используемые программистом, однако не приведет к созданию определенных фрагментов кода, которые активизируют эту возможность. Другие же атрибуты качества действительно реализуются в коде. Требования к целостности для аутентификации пользователей инициируют создание производных функциональных требований, которые реализуются, с помощью, скажем, паролей или биометрических параметров. В этих случаях следует отслеживать соответствующие функциональные требования в обратном направлении, к их родительским нефункциональным требованиям, и, как обычно, в прямом, до готового продукта. На рис. 29-3 показана возможная цепь отслеживаемости с участием нефункциональных требований.

Связи отслеживаемости должны определять любое лицо, имеющее доступ к соответствующей информации. В табл. 29-3 определены некоторые стандартные источники информации о связях между различными типами исходных и целевых объектов. Определите роли и лиц, которые будут поддерживать каждый тип информации отслеживаемости для вашего проекта. Будьте готовы к тому, что занятые люди, которых бизнес-аналитик или менеджер проекта попросит предоставить эти данные, будут отнекиваться. Этим

специалистам стоит объяснить, что такое отслеживание требований, чем оно ценно и почему именно этих специалистов просят внести свой вклад в процесс. Подчеркните, что увеличение затрат на фиксирование информации отслеживаемости во время выполнения невелики; в основном это вопрос привычки и дисциплины, а также наличия установленного механизма хранения.

> **Внимание!** Нужно назначить конкретных лиц, ответственных за сбор и управление данными отслеживаемости требований, иначе это просто не будет сделано. Обычно бизнес-аналитик или специалист по проверке соответствия качества собирает, сохраняет информацию такого рода и составляет отчеты по информации отслеживаемости.

Табл. 29-3. Возможные источники информации о связях отслеживаемости

Тип объекта источника ссылки	Тип объекта целевой ссылки	Источник информации
Системное требование	Функциональное требование	Системный инженер
Пользовательское требование	Функциональное требование	Бизнес-аналитик
Бизнес-требование	Пользовательское требование	Бизнес-аналитик
Функциональное требование	Функциональное требование	Бизнес-аналитик
Функциональное требование	Тест	Тестировщик
Функциональное требование	Элемент архитектуры	Архитектор или разработчик
Функциональное требование	Другие элементы дизайна	Дизайнер или разработчик
Элемент дизайна	Код	Разработчик
Бизнес-правило	Функциональное требование	Бизнес-аналитик

Средства отслеживания требований

В главе 21 описано несколько серийных средств управления требованиями, обладающих мощными возможностями отслеживания требований. Вы можете сохранить требования и другую информацию в базе данных средства и определить связи между различными типами сохраненных объектов, включая равноценные ссылки между двумя требованиями одного типа. Некоторые инструменты позволят вам различить отношения «отслеживается до» и «отслеживается от», автоматически определяя дополнительную ссылку. То есть, если вы укажете, что требование R отслежено до теста T, средство также покажет симметричное отношение, в котором T отслеживается от R.

Некоторые средства автоматически каждый раз помечают ссылку как *подозрительную*, когда объект на одном из концов связи изменяется. Подозритель-

ная ссылка помечается (например, знаком вопроса красного цвета или диагональной линией красного цвета) в соответствующей ячейке матрицы отслеживания требований. Например, если вы измените вариант использования 3, то матрица отслеживания требований в табл. 29-2 может выглядеть, как показанная в табл. 29-4, когда вы на нее посмотрите в следующий раз. Указатель подозрительной связи (в данном случае знак вопроса) напоминает вам, что нужно проверить, следует ли модифицировать функциональные требования 3, 4 и 6, чтобы те остались совместимыми с измененным UC-3. После внесения всех необходимых изменений необходимо вручную убрать указатели подозрительных ссылок. Это процесс помогает убедиться, что вы учли все возникшие в результате «эффекта кругов на воде» изменения.

Табл. 29-4. Подозрительные связи в матрице отслеживания требований

	Вариант использования			
Функциональное требование	**UC-1**	**UC-2**	**UC-3**	**UC-4**
FR-1	←┘			
FR-2	←┘			
FR-3			←?┘	
FR-4			←?┘	
FR-5		←┘		←┘
FR-6			←?┘	

Средства управления требованиями также позволяют определить перекрестные связи между проектами или между подсистемами. Я знаю об одном крупном проекте ПО с 20 крупными подсистемами, где требования к продукту высокого уровня распределены среди этих подсистем. В некоторых случаях требование, адресованное одной подсистеме, реализовалось с помощью службы, предоставленной другой подсистемой. В этом проекте для успешного отслеживания этих сложных отношений использовалось средство управления требованиями.

Отслеживание требований вручную невозможно выполнить ни для одного приложения, за исключением самых маленьких. Вы можете воспользоваться рабочей таблицей для поддержания данных отслеживаемости пары сотен требований, однако для более крупных систем необходимо решение понадежнее. Отслеживание требований нельзя полностью автоматизировать, поскольку данные об источниках связей хранятся в головах разработчиков. Однако после того как вы определите связи, средства помогут вам управлять огромным объемом информации отслеживаемости.

Процедура отслеживания требований

Придерживайтесь следующей последовательности действий при реализации отслеживания требований в проекте.

1. Проинформируйте команду о принципах и важности отслеживания требований, о целях этого, о том, где хранятся данные отслеживаемости, и приемах для определения связей. Убедитесь, что все участники осознают их ответственность.
2. Выберите отношения связей, которое вы хотите определить, из тех, что показаны на рис. 29-2. Не пытайтесь использовать их все за раз — зашьетесь с работой!
3. Выберите тип матрицы отслеживания требований, которую вы хотите использовать: одну матрицу, показанную в табл. 29-1, или несколько матриц, показанных в табл. 29-2. Выберите механизм для хранения данных: таблица в текстовом документе, рабочая таблица или средство управления требованиями.
4. Определите части продукта, для которых вы планируете поддерживать информацию отслеживаемости. Начните с важнейших основополагающих функций или частей, которые, как вы и ожидали, потребуют большей части поддержки и будут быстрее развиваться в течение жизненного цикла продукта.
5. Определите лиц, которые будут поддерживать отдельные типы информации о связях, и сотрудника (скорее бизнес-аналитика), который будет координировать отслеживание и управление данными.
6. Модифицируйте процедуры разработки и списки вопросов, чтобы напомнить разработчикам об обновлении связей после реализации требования или утверждения изменения. Данные отслеживаемости должны обновляться сразу после завершения задачи, в результате которой создается или изменяется звено в цепи требований.
7. Определите соглашения об именовании, которые вы будете использовать для уникальной идентификации всех элементов системы таким образом, чтобы их удалось связать друг с другом. Подробнее о различных способах именования требований см. главу 10.
8. Попросите каждого участника предоставлять запрашиваемую информацию отслеживаемости по мере завершения небольших участков работы. Напоминайте о том, что работа над созданием данных отслеживаемости не прекращается ни на минуту; не говорите о том, что вы попытаетесь воссоздать эти данные на важном этапе работы или в конце проекта.
9. Периодически проверяйте информацию отслеживаемости, чтобы убедиться, что они не устарели. Если поступает сообщение, что требование реализовано и проверено, но данные отслеживаемости неполны или неточны, значит, что процесс отслеживания требований не работает так, как ожидалось.

Я описал эту процедуру, как если бы вы начинали собирать информацию отслеживаемости с начала нового проекта. Если вы обслуживаете действующую систему, то бьюсь об заклад, что у вас нет доступных данных отслеживаемости, однако у вас нет времени начинать сбор этой полезной информации. Но самое

лучшее время для этого — прямо сейчас. В следующий раз, когда вы добавите улучшение или внесете изменение, запишите все, что вы узнали в связях между кодом, тестами, дизайном и требованиями. Вам уже не удастся собрать полную матрицу отслеживания требований, однако это небольшое усилие может облегчить работу над этой же частью системы в следующий раз.

К этому хорошо подойдет кофе и музыка

Моя знакомая по имени Соноко, очень опытный разработчик ПО, работающий над системами обработки транзакций с кредитными картами, недавно прислала мне сообщение электронной почты. «Я думаю, тебе будет интересно узнать, что я потратила вторую часть дня на создание матрицы отслеживания требований для одного из моих проектов, и я буквально умираю со скуки, — написала Соноко. — Спецификация требований состоит из 30 страниц, мой технический дизайн занимает 100 страниц, и матрица получилась немаленькая. Я знаю, что нам нужно их делать, но я уже два часа как нахожусь в летаргическом состоянии».

Я задал Соноко несколько вопросов вдогонку, чтобы лучше понять, что происходит. На что она ответила: «Так как я предоставила свой технический дизайн бизнес-аналитику, заинтересованным бизнес-подразделениям и менеджеру проекта, матрица отслеживаемости доказала им, что я учла все требования, которые они мне дали. В своей рецензии дизайна я представила дизайн, пройдясь по матрице отслеживаемости, за которой логически следует требование». Я спросил Соноко, почему она тратит время на создание матрицы отслеживаемости, на что она ответила: «Я делаю ее, потому что она гарантирует, что я ничего не пропустила, а также позволяет мне быстро увидеть все элементы системы, на которые влияет то или иное требование».

Не один десяток лет занимаясь разработкой ПО, Соноко отлично понимает ценность связывания требования с элементами дизайна, на которые оно влияет. Но, как она и говорит, продираться сквозь огромный объем информации и увязывать мелкие части вместе — занятие не самое веселое. Если позволяет ситуация, она экономит время, начав собирать информацию о связях не в конце, а когда дизайн переходит на стадию стабилизации.

Осуществимость и необходимость отслеживания требований

Вы можете прийти к заключению, что на создание матрицы отслеживания требований вы тратите больше, чем получаете, или что она неосуществима для вашего крупного проекта, поэтому я хочу уверить вас, что это не так. Приобретение средства с соответствующей функциональностью, настройка, ввод данных и поддержка в актуальном состоянии дороги и требуют време-

ни. Может быть, что вам не нужно создавать подобную «групповую память», если члены вашей команды обладают соответствующим знанием и делятся им с другими по мере необходимости. Только ваша команда может решить, будет ли польза для проекта от отслеживания требований — будь то простое отслеживание связей между требованиями и тестами или что-то более сложное.

Приведу один пример. Как-то на конференции один участник, занимающийся строительством самолетов, сказал мне, что спецификация требований для работы над последним реактивным самолетом представляла собой кипу бумаги высотой в шесть футов (примерно два метра — *прим. перев.*). У них была полная матрица отслеживания требований. Я летал именно на этой модели самолета и был счастлив узнать, что разработчики так тщательно поработали над своими требованиями к ПО. Управление отслеживаемостью для большого продукта с множеством взаимосвязанных подсистем — это большой объем работы. Мой новый знакомый знал, что оно очень важно, и Федеральное управление гражданской авиации было с ним согласно: отслеживаемость требований необходима для сертификации авиационного ПО. Аналогично Администрация по контролю за продуктами питания и лекарствами США требует от производителей медицинских приборов, чтобы требования к продукту отслеживались в результатах, — это входит в стандартную процедуру приемки устройства.

Даже если в случае неудачи ваши продукты не могут стать причиной угрозы жизни или здоровью людей, вам следует серьезно отнестись к отслеживанию требований. Как минимум стоит отслеживать связи между бизнес-требованиями и пользовательскими требованиями на предмет соответствия, пропусков или ненужных требований. Когда я рассказывал об отслеживаемости на семинаре, глава крупной корпорации спросил: «Каковы могут быть причины *не создавать* матрицу для отслеживания требований к вашей стратегической бизнес-системе?» Это отличный вопрос. Вы должны принимать решение об использовании любых улучшенных приемов разработки требований, принимая во внимание и затраты на внедрение приема, и риск *не применения* приема. Как и со всеми процессами ПО, экономические факторы должны определять, тратить ли ваше ценное время там, где возможна наибольшая отдача.

Что дальше?

- Создайте матрицу отслеживания требований для 15 или 20 требований к важной части системы, которую вы в настоящее время разрабатываете. Опробуйте приемы, показанные в табл. 29-1 и в табл. 29-2. Заполняйте матрицу по мере разработки, в течение нескольких недель. Оцените, какой метод окажется наиболее эффективным и какие процедуры для сбора и хранения информации отслеживаемости подойдут для вашей команды.
- В следующий раз, когда будете обслуживать плохо задокументированную систему, записывайте, чему научил вас обратный инженерный

анализ продукта, который вы модифицируете. Постройте фрагмент матрицы отслеживания требований для фрагмента головоломки, над которой вы сейчас работаете, чтобы облегчить жизнь тому, кто займется ей в следующий раз. Расширяйте матрицу отслеживания требований по мере обслуживания проекта.

- Отследите свои функциональные требования обратно к пользовательским требованиям и пользовательские требования — к функциональным. Посчитайте число требований, от которых можно избавиться, потому что они не ссылаются обратно на какое-либо бизнес-требование. Посчитайте, сколько отсутствующих требований было выявлено с помощью матрицы отслеживания. Оцените возможные потери, если бы вы обнаружили ошибки в этих требованиях на более поздних стадиях проекта. Этот анализ позволит оценить, стоит ли отслеживать требования в вашей среде.

Глава 30

Инструментальные средства разработки требований

Эстель наконец добилась завершения и утверждения своего документа спецификации требований к ПО. И тут Джеймс захотел добавить требование, но оно нарушает схему нумерации и требуется увеличить номера требований в соответствующем разделе документа. Эстель надеется, что изменение идентификаторов требований не создаст проблем всем тем, кто уже использует эти требованиям в своей работе. Шон попросил удалить одно требование. Эстель подозревает, что через некоторое время требование вернется, поэтому она думает, где разместить его и как сделать так, чтобы разработчики не начали над ним работать. Вчера Антонио спросил Эстель о причине включения одного требованиям, но она не смогла ответить на этот вопрос.

Один из разработчиков по имени Рам попросил представить список всех требований, за реализацию которых в следующем выпуске он отвечает, но у Эстель нет готового способа создать такой список. На самом деле, очень непросто отследить, какие требования назначены на тот или иной выпуск, потому что они находятся в одном документе. Эстель хотела бы знать состояние требований, которые уже ушли в разработку, но и эту информацию получить непросто.

Используемый Эстель основанный на документе подход к работе с требованиями не удовлетворяет потребностей по управлению требованиями. Ей нужно специальное инструментальное средство.

В предыдущих главах я уже говорил о создании спецификации требований к ПО на естественном языке, которая содержит функциональные и нефункциональные требования, а также, о создании документов, содержащих бизнес-требования и описания вариантов использования. Мы говорили, что эти результаты представляют собой всего лишь контейнеры для хранения совокупности информации о требованиях и не обязательно должны существовать в форме документов, созданных в текстовом редакторе. Документальный способ разработки и управления требованиями используется очень широко, но у него много ограничений, в том числе:

- трудность обновления и синхронизации документов;

- доведение изменений до всех членов команды, которым это необходимо, приходится осуществлять вручную;
- трудность хранения дополнительной информации (атрибутов) для каждого требования;
- трудность определения взаимосвязей между требованиями и другими элементами системы;
- отслеживание состояния отдельных требований и их наборов представляет собой трудный и неудобный процесс;
- одновременное управление наборами требований, запланированных для различных выпусков или взаимосвязанных продуктов, затруднено. Когда реализация требования откладывается до следующего выпуска, аналитику приходится перемещать его из одной спецификации требований в другую;
- при повторном использовании какого-то требования аналитику приходится многократно копировать текст исходной спецификации требований в спецификацию требований каждой системы или продукта, где оно будет использоваться;
- трудность модификации требований несколькими участниками проекта, особенно если они географически разделены;
- отсутствие удобного места хранения требований, которые были предложены, но отклонены, или требований, удаленных из базовой версии;
- тяжело создавать и отслеживать связи исправлений и моделей анализа в том же месте, что и требования;
- сложность обнаружения пропущенных, дублирующихся или ненужных требований.

Средство разработки и управления требованиями позволяет снять эти ограничения. Средства разработки требований помогают выявить правильные требования для проекта и оценить качество написания этих требований. Средства управления требованиями помогают управлять изменениями этих требований и отслеживать их связи с другими результатами проекта.

В небольших проектах можно использовать для управления требованиями электронные таблицы или простые базы данных. В более крупных проектах выгодно применять серийные средства управления требованиями. Эти инструментальные средства не заменяют собой приемов, с помощью которых члены вашей команды выявляют требования и управляют ими. Используйте инструменты, когда у вас уже есть рабочая методика, но ей не хватает эффективности; не надейтесь, что инструмент восполнит недостаток трудолюбия, дисциплины, опыта или понимания.

> **Внимание!** Избегайте искушения разработать собственное средство управления требованиями или на скорую руку слепить его из средств офисной автоматизации общего назначения в подражание коммерческим продуктам. Первоначально это кажется легким решением, но такая работа может быстро истощить ресурсы команды, не обладающей средствами, необходимыми для создания желаемого инструмента.

Данная глава рассказывает о нескольких преимуществах использования средств управления требованиями и указывает на некоторые общие возможности таких продуктов. На рынке сейчас доступно несколько десятков инструментальных средств управления требованиями. В этой главе я не буду подробное сравнивать их — функция за функцией, поскольку эти продукты (и поставщики) и их возможности постоянно меняются. Эти средства не дешевы, но высокая стоимость решения проблем управления требованиями оправдывает затраты на них. Учитывайте, что цена инструментального средства определяется не только ценой лицензии. Вам придется потратиться на компьютер, на который будет установлена программа, учесть ежегодные затраты на поддержку и периодическое обновление продукта, на установку ПО, администрирование, техническую поддержку и консультации производителя, а также на обучение пользователей. В облачных решениях нет части из перечисленных статей расходов. Проанализируйте возможные затраты и результаты до того, как примете решение о покупке.

Средства разработки требований

Средства разработки требований используются бизнес-аналитиками при работе с заинтересованными лицами в процессе выявления и документирования требований, обеспечивая большую производительность и эффективность этих операций. Заинтересованные лица по-разному потребляют и обмениваются информацией: в текстовой, визуальной или звуковой форме. Средства разработки требований могут повысить уровень сотрудничества заинтересованных лиц, поддерживая разные методы коммуникации (Frye, 2009). В этом разделе инструменты разработки разбиваются на средства выявления требований, создания прототипов и моделирования. Некоторые средства из категории разработки требований поддерживают все эти возможности. Некоторые из них обеспечивают функции управления требованиями. В целом средства разработки требований не такие зрелые, как инструменты управления требованиями, и их влияние на проекты обычно не такое большое, как влияние последних.

Средства выявления требований

К средствам выявления требований относятся те, что применяются для создания заметок в процессе встреч по выявлению требований. Они позволяют бизнес-аналитику быстро организовывать идеи и формулировать вопросы, планируемые действия, основные термины и т. п. Средства планирования облегчают проведение мозговых штурмов, а также упорядочение собранной информации. Аудио-ручки и другие средства записи позволяют в дальнейшем воспроизвести разговор или дают визуальные напоминания того, что происходило в процессе встречи по выявлению требований. Некоторые записывающие устройства также позволяют привязывать аудио к записываемому в соответствующий момент тексту, позволяя по мере необходимости

прослушивать конкретные отрывки. Инструменты, поддерживающие проверки качества, например сканирование документа требований на предмет туманных и двузначных формулировок, помогают бизнес-аналитику писать более четкие требования. Некоторые средства выявления требований преобразуют требования из текста в автоматически генерируемые диаграммы. Некоторые средства поддерживают голосование внутри команды для определения приоритетов требований.

Средства создания прототипов

Средства создания прототипов применяются для создания рабочих продуктов, диапазон которых распространяется от электронных макетов до полноценной имитации приложений. Простые средства создания прототипов поддерживают базовые образы и дизайн с возможностью создания простых каркасов (Garmahis, 2009). Популярные приложения, такие как Microsoft PowerPoint, можно использовать для быстрого создания макетов экранов и навигации между ними или для аннотирования имеющихся снимков экрана. Сложные средства позволяют создавать имитации функциональности, которая позволяет пользователю увидеть, как работает приложение. Некоторые средства создания прототипов поддерживают управление версиями и обратной связью, создание связей требований и генерацию кода. Подробнее об опасности больших затрат на создание прототипов, чем это нужно для дела, см. главу 15. Если вы используете инструментальное средство для создания высококачественных прототипов, четко объясните клиентам, что прототипы представляют собой всего лишь возможные модели, а конечный продукт может отличаться. Некоторые средства создания прототипов позволяют отображать модели «нарисованных от руки» экранов, которые позволяют управлять ожиданиями клиентов.

Средства моделирования

Средства моделирования требований помогают бизнес-аналитикам создавать диаграммы, подобные тем, которые описаны в главах 5, 12 и 13. Эти средства поддерживают использование стандартных форм, нотации и синтаксиса для рисования диаграмм в соответствии с установленными соглашениями. Они могут предоставлять в качестве исходной точки шаблоны и примеры, которые позволяют бизнес-аналитику узнать больше о конкретной модели. Часто эти средства автоматически объединяют формы в диаграммах для ускорения процесса рисования и для обеспечения правильности рисования диаграмм. Они также позволяют вам создавать диаграммы, которые выглядят более наглядными и единообразными, чем если бы вы рисовали их вручную. Специализированные средства моделирования ПО поддерживают итеративный подход за счет перемещения всех присоединенных стрелок и меток вместе с перемещением символа в диаграмме; средства общего назначения могут не поддерживать этой функциональности.

Многие средства управления требованиями поддерживают некоторые возможности по моделированию. Самые развитые средства позволяют отслеживать связи отдельных требований к моделям или даже конкретным элементам моделей. Например, аналитики могут создавать в таком инструменте swimlane-диаграммы, а затем, после написания требования, отслеживать эти требования по конкретным шагам в диаграммах.

Помните, что никакое средство не позволит узнать, не пропущено ли какое-то требование или элемент модели, не является ли оно логически неверным или ненужным. Эти средства позволяют бизнес-аналитикам представлять информацию в различных видах и обнаруживать определенные типы ошибок и пропусков, но они не избавляют от необходимости думать и выполнять процедуру рецензирования.

Средства управления требованиями

Средство управления требованиями, хранящее информацию в многопользовательской базе данных, позволяет снять ограничения на хранение требований в документах. В небольших проектах можно ограничиться вводом текста требования и несколькими атрибутами каждого требования. В более крупных проектах выгодны возможности, позволяющие пользователям импортировать требования из исходных документов, определять значения атрибутов, фильтровать и выводить на экран содержание базы данных, экспортировать требования в различных форматах, контролировать связи отслеживания и соединять требования с элементами, хранящимися в других средствах разработки ПО.

Средствам управления требованиями уже много лет. Их больше и они намного более зрелые, чем средства разработки требований. Справедливости ради нужно сказать, что решаемая ими проблема проще. Существенно проще создать базу данных для хранения требований и предоставить возможности манипуляции с ними, чем помочь бизнес-аналитику обнаруживать новые знания, преобразовывать его в точные формулировки требований и диаграммы и обеспечивать корректность полученного в результате представления информации. В некоторых, самых мощных средствах объединены возможности управления и разработки требований.

Преимущества использования средств управления требованиями

Даже если вы провели внушительную работу по сбору и спецификации требований для своего проекта, в процессе разработки вы можете потерять контроль над ними. Инструментальное средство управления требованиями становится все полезнее со временем, когда детали требований постепенно тускнеют в памяти членов команды. В следующих разделах описаны некоторые задачи, которые подобное средство поможет вам решать.

Управление версиями и изменениями Ваш проект должен определять основную версию требований — четко определенный набор требований для конкретной версии или итерации. Некоторые средства управления требованиями поддерживают функции гибкого управления базовой версией. Они также сохраняют историю изменений каждого требования. Вы можете записывать обоснование каждого решения об изменении и при необходимости возвратиться к предыдущей версии требования. Некоторые средства системы предложения изменений, устанавливающие связи между предложениями об изменениях и измененными требованиями.

Хранение атрибутов требований Вы должны записывать несколько описательных атрибутов для каждого требования, как говорилось в главе 27. Каждый, кто работает над проектом, должен иметь доступ к просмотру этих атрибутов, а некоторые — к изменению их значений. Средства управления требованиями генерируют несколько системных атрибутов, например дату создания требования и номер его версии, а также, позволяют создавать дополнительные атрибуты различных типов. Продуманное определение атрибутов позволяет всем заинтересованным в проекте лицам просматривать подмножества требований, основанных на выбранных комбинациях значений атрибутов. Один из способов учета требований в основных версиях различных выпусков продукта — использовать атрибут «номер выпуска».

Способствование анализу воздействия Средства управления требованиями помогают отслеживать требования, давая возможность определять связи между различными типами требований, между требованиями в различных подсистемах и между отдельными требованиями и связанными системными компонентами (например, дизайном, модулями кода, тестами и пользовательской документацией). Эти связи помогают анализировать воздействие, которое предлагаемое изменение окажет на конкретное требование, выявляя другие элементы системы, которые оно затронет. Другой полезный способ отследить каждое функциональное требование обратно до первоисточника, чтобы знать, откуда оно берет начало. Например, вы можете запросить список всех требований, основанных на каком-либо бизнес-правиле, чтобы принять решение о последствиях изменения этого правила. В главе 28 описан анализ воздействия, а в главе 29 — отслеживание требований.

Выявление отсутствующих и излишних требований Функциональность отслеживания в средствах управления требованиями помогает заинтересованным лицам выявлять отсутствующие требования, например пользовательские требованиям, у которых нет соответствующих функциональных требований. Также она позволит выявить требования, у которых нет разумного источника, что вызывает вопрос о том, нужны ли такие требования вообще. При исключении бизнес-требования можно быстро удалить все связанные с ним требования.

Отслеживание состояния требований Собрав требования в базе данных, вы узнаете, сколько отдельных требований определено для продукта. Как описано в главе 27, отслеживание состояния каждого требования в процессе разработки способствует отслеживанию состояния проекта в целом.

Управление доступом Средства управления требованиями позволяют устанавливать права доступа для отдельных людей и групп пользователей и предоставлять информацию в общий доступ географически распределенной команде через веб-интерфейс к базе данных. Некоторые средства позволяют сразу многим пользователям обновлять содержимое базы данных одновременно.

Связь со всеми заинтересованными в проекте лицами Средство управления требованиями служит главным хранилищем, позволяя членам команды работать с одним и тем же набором требований. Некоторые средства управления требованиями позволяют членам команды обсуждать вопросы, связанные с требованиями, на тематических дискуссиях с помощью электронных средств обмена сообщениями. Автоматически отсылаемые электронные сообщения уведомляют членов команды, когда в дискуссии появляется новая запись или любое требование модифицируется. Возможность работы с требованиями через Интернет сокращает расходы на транспорт и уменьшает документооборот.

Повторное использование требований Хранение требований в базе данных облегчает их повторное использование в других проектах и подпроектах. Требования, логически подходящие нескольким разделам описания проекта, можно сохранить однажды, а затем лишь ссылаться на них во избежание дублирования требований. Подробнее о принципах эффективного повторного использования требований см. главу 18.

Отслеживание состояния дефектов Некоторые средства управления требованиями позволяют отслеживать открытие дефекты и связывать их с соответствующими требованиями. При устранении дефекта легко определить, нужно ли обновить соответствующие требования. Вы также можете легко просмотреть историю дефекта и его устранения. Отслеживание дефектов в специальном средстве позволяет автоматически получать отчеты о ситуации с дефектами.

Генерирование произвольных подмножеств Средства управления требованиями позволяют извлечь и просмотреть набор требований для определенной цели. Например, может потребоваться получить отчет, содержащий все требования для определенной итерации разработки, все требования, относящиеся к определенной функции или набор требований, нуждающихся в проверке.

Возможности средств управления требованиями

Дерево возможностей на рис. 30-1 представляет сводку возможностей, которые обычно имеются в средствах управления требованиями. Подробный сравнительный анализ возможностей различных средств управления требованиями можно найти в Интернете (например, см. Seilevel, 2011; INCOSE, 2010; Volere, 2013).

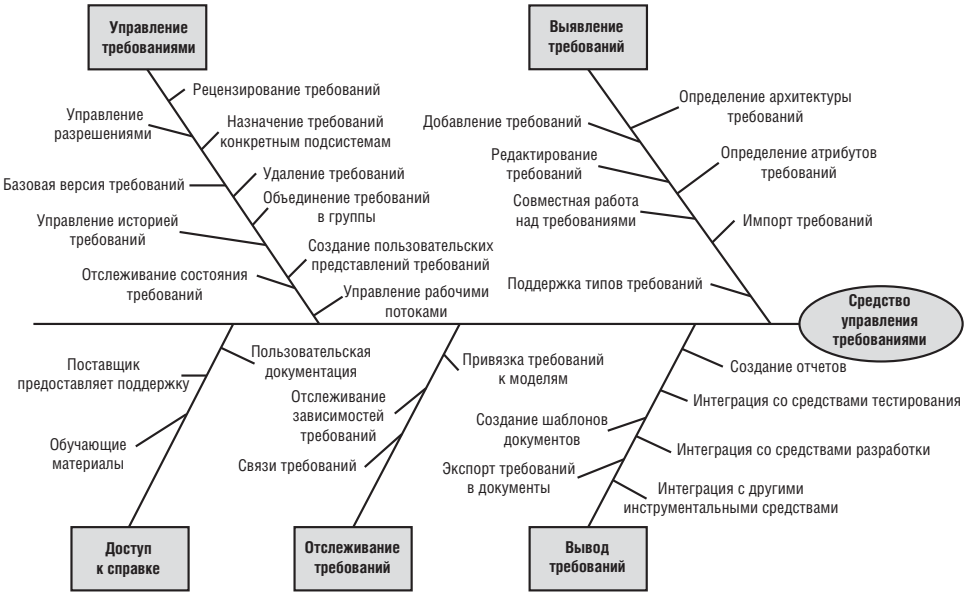


Рис. 30-1. Стандартные функции средства управления требованиями

Коммерческие средства управления требованиями позволяют определять различные типы требований, такие, как бизнес-требования, варианты использования, функциональные требования, требования к оборудованию, а также ограничения. Это позволяет вам отделять задачи, с которыми можно работать, как с требованиями, от другой полезной информации, содержащейся в спецификации требований к ПО. Многие средства позволяют конфигурировать архитектуру информации (она определяет, как связаны типы требований с другими объектами) в соответствии с принятыми в команде приемами работы. Подробнее о стандартных связях, которые можно определять в информационной архитектуре см. главу 29. Большинство инструментальных средств обладают мощными возможностями определения атрибутов для каждого типа требований, что представляет огромное преимущество перед обычными способами документирования спецификации требований к ПО.

Средства управления требованиями обычно поддерживают иерархическую систему нумерации требований помимо уникального внутреннего идентификатора для каждого требования. Эти идентификаторы обычно состоят из короткого текстового префикса, который указывает на тип требования, например UR обозначает требование пользователя (user requirement), и уникального целого числа. Некоторые средства предоставляют возможность управления иерархическим деревом требований.

Требования могут импортироваться в средство управления требованиями в различных популярных форматах. Как правило, текстовое описание требования считается просто одним из обязательных атрибутов. Некоторые продукты позволяют включать в базу данных нетекстовые объекты, такие как графика и электронные таблицы. Другие продукты позволяют устанавли-

вать связи отдельных требований с внешними файлами (например, файлами Microsoft Word, Microsoft Excel, графическими файлами и т. д.), в которых содержится информация, дополняющая содержимое базы.

К возможностям вывода данных инструментальных средств относится способность генерировать требования либо в документе заданного пользователем формата, в табличном отчете или в виде веб-страницы. Некоторые средства позволяют определять шаблоны, в которых задается структура страницы, шаблонный текст, извлекаемые из базы данных атрибуты и стили текста. Таким образом, спецификация требований к ПО, в сущности, представляет собой отчет, генерируемый по некоторой выборке из содержимого базы данных по определенным критериям и форматируемый, как обычная спецификация. Например, можно создать спецификацию требований к ПО, которая содержит все функциональные требования, назначенные на определенный выпуск и конкретному разработчику. Некоторые средства предоставляют пользователям функциональность внесения изменений в экспортированные документы в автономном режиме, а после перехода в интерактивный режим эти документы синхронизируются с базой данных.

Большинство инструментов поддерживают различные представления требований, которые создаются в самом средстве или экспортируются из него. К числу других функций относится возможность задавать группы пользователей и определять разрешения некоторым пользователям или группам на создание, чтение, обновление и удаление проектов, требований, атрибутов и их значений. Настройка соответствующих представлений и разрешений позволяет контролировать требования и процессы коллективной работы в процессе улучшения требований. Некоторые инструменты предлагают также средства обучения или проекты-примеры, которые помогут пользователям научиться работать быстро и эффективно.

Средства обычно обладают мощными возможностями отслеживания. Отслеживание реализуется за счет определения связей между объектами двух типов или внутри одного типа. Некоторые средства управления требованиями включают возможности моделирования, которые тоже позволяют связывать модели на уровне элементов с конкретными требованиями или другими элементами модели.

Некоторые средства управления проектами гибкой разработки также предоставляют возможности управления требованиями. Они используются для управления и приоритизации резерва проекта, назначения требований конкретным итерациям и генерации тестов непосредственно на основе требований.

Средства управления требованиями часто интегрируются с другими инструментами, применяемыми при разработке приложений, как показано на рис. 30-2. Подробнее о связывании требований с объектами, находящимися в других инструментах, см. главу 29. Например, можно отслеживать связи конкретных требований с отдельными элементами дизайна, хранящимися в средстве моделирования дизайна, или тестами, находящимися в средстве управления тестированием.

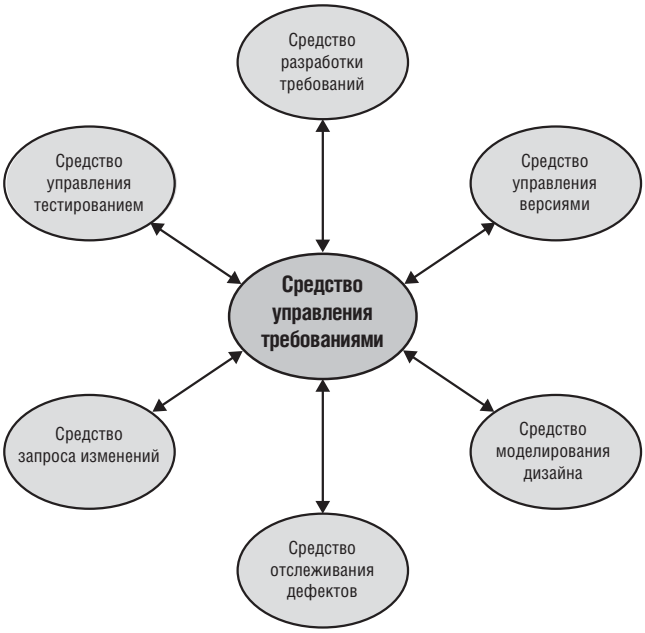


Рис. 30-2. Средства управления требованиями интегрируются с другими видами средств разработки

Оценивая средство управления требованиями, выясните, поддерживает ли оно обмен данными с другими используемыми вами средствами. Подумайте, как вы сможете воспользоваться преимуществом интеграции продуктов при составлении требований, тестировании, отслеживании связей проекта и других процессах. Например, посмотрите, как определяются связи между функциональными требованиями и конкретными элементами дизайна или кода или как убедиться, что выполнены все тесты, связанные с конкретными функциональными требованиями.

Выбор и реализация средства управления требованиями

Любой из этих продуктов позволяет управлять требованиями на более высоком и производительном уровне. Тем не менее, успех зависит от выбора средства, которое больше всего подходит вашей организации, и внедрения его в ежедневную работу вашей команды.

Выбор инструментального средства

Выбирайте инструментальное средство, учитывая такие параметры, как требуемые функции, платформа и цена, которые лучше всего подходит вашей среде разработки и культуре. Бизнес-аналитики должны возглавить деятельность оп определению критериев оценки и по проведению самой оценки.

Некоторые компании привлекают для оценки инструментальных средств консультантов, которые могут оценить все нужды компании и порекомендовать доступный продукт. Если вы проводите оценку самостоятельно, приведенные в главе 22 рекомендации по выбору готового серийного решения помогут вам сделать правильный выбор. В главе также приводится реальный пример оценки одного средства для работы с требованиями. Вот сводка рекомендаций по процессу выбора:

1. Определите требования организации к средству управления требованиями, основываясь на критериях оценки.
2. Определите приоритеты и веса критериев в соответствии с важностью возможностей и других обстоятельств для вашей организации.
3. Установите демо-версии или приобретите ознакомительные копии средств, из которых планируется сделать выбор.
4. Используя единообразную процедуру, определите баллы соответствия каждого средства критериям.
5. Вычислите общий бал для каждого средства на основе баллов и веса для каждого критерия.
6. Каждое из средств, получивших максимальное число баллом, используйте в реальном проекте, чтобы посмотреть, ведет ли оно себя именно так, как ожидалось на основе объективной оценки.
7. В завершение примите решение, учитывая как количество баллов, набранных каждым продуктом, стоимость лицензий и расходов на сопровождение, так мнение других пользователей и субъективные впечатления членов вашей команды от каждого продукта. Есть два очень хороших вопроса, позволяющих выполнить окончательную оценку средств: «Какое средство вы хотели бы использовать?» и «Использование какого средства вас больше всего напрягало, если бы его выбрали и вам пришлось его использовать?

Настройка средств и процессов

Учитывайте, что загрузка требований проекта в базу данных, определение атрибутов и связей отслеживания, своевременное обновление содержимого базы данных, определение доступа групп и их привилегий, а также подготовка пользователей потребуют усилий. Конфигурирование средства может оказаться сложной задачей; при настройке сложного средства для работы с требованиями нужно в короткое время усвоить большой объем материала. Руководство должно выделить ресурсы, необходимые для этих операций. Доведите до сведения всех сотрудников вашей фирмы просьбу действительно использовать выбранный продукт, иначе оно просто превратится в дорогое украшение, которое лежит на полке.

Нет большого смысла применять инструментальное средство управления требованиями, если вы не воспользуетесь его возможностями. Я знал коман-

ду, которая, работая над проектом, прилежно сохраняла все требования в таком средстве, но не определяла для этих требований никаких атрибутов или связей отслеживания. Не предоставили они и интерактивного доступа заинтересованным в проекте лицам. Тот факт, что требования хранились в иной форме, не дал существенных преимуществ, хотя было затрачено много сил для ввода требований в программу. Другая команда хранила сотни требований в инструментальном средстве и установила много связей для отслеживания. Но единственное, для чего использовались эти данные, была генерация массы предназначенных для печати отчетов отслеживания, которые потом предполагалось вручную проверять на предмет ошибок. Никто эти отчеты не изучал, и никто не считал базу данных полномочным хранилищем требований. Ни одна из этих организаций не получила полной выгоды от существенных инвестиций времени и денег в средства управления требованиями.

Даже если вы выберете самое лучшее средство из имеющихся, оно не обязательно даст все нужные вашей организации возможности. Оно может не поддерживать существующие у вас шаблоны и процессы работы с требованиями. Вам скорее всего придется адаптировать некоторые из существующих процессов для внедрения в них этого средства. Рассчитывайте на то, что придется вносить изменения в шаблоны, имена атрибутов и порядок действий по разработке требований. Воспользуйтесь следующими рекомендациями, если хотите получить максимальную выгоду от вложений в серийные средства управления требованиями:

- назначьте опытного бизнес-аналитика ответственным за настройку и адаптацию процессов. Он сможет понять влияние выбора конфигурации и изменений процессов;
- тщательно продумывайте, какие типы требований вы определяете. Не считайте каждый раздел вашей нынешней спецификации требований отдельным типом требований, но и не помещайте все содержимое спецификации в единственный тип требований.
- используйте инструмент как средство налаживания связей между заинтересованными в проекте лицами, которые находятся в различных местах. Настройте доступ и измените привилегии различным людям, чтобы позволить им ввод данных в требования, не давая при этом им права изменять всю базу данных;
- не пытайтесь вносить требования непосредственно в инструментальное средство, когда только начинаете проводить семинары по выявлению требований. Однако по мере того как требования начнут обретать устойчивые формы, хранение их в программе откроет доступ к ним участникам семинаров для уточнения;
- используйте средства разработки требований во время их выявления, только если уверены, что они не замедлят процесс обнаружения и не приведут к пустой трате времени заинтересованными лицами;

- не определяйте связей отслеживания, пока требования не обретут устойчивые формы. В противном случае вам придется исправлять массу связей по мере изменения требований;
- для ускорения перехода от документального способа хранения данных к использованию набора инструментальных средств, назначьте дату, после которой база данных инструментального средства будет считаться окончательным хранилищем требований проекта. После этой даты все требования, содержащиеся в формате текстовых редакторов, не будут считаться документами, имеющими силу.

Если вы не забыли, что инструментальное средство не может исправить недостатки процесса, то скорее всего согласитесь с тем, что коммерческие средства управления требованиями помогают вам управлять требованиями к ПО.

> **Внимание!** Даже не пытайтесь использовать инструментальное средство, пока ваша организация не сможет создать приемлемую спецификацию требований к ПО на бумаге. Если ваша основная проблема связана со сбором и записью ясных и качественных требований, средство управления требованиями вам не поможет (а вот средство разработки требований может помочь).

Освоение средств пользователями

Желание пользователей инструментальных средств остается критическим фактором успеха. Усердные, дисциплинированные и знающие люди добиваются успехов, даже работая с посредственным инструментарием, тогда как самые лучшие средства не оправдают даже собственной стоимости в руках пользователей, не обладающих должной мотивацией или подготовкой. Не выписывайте чек на оплату средства управления требованиями, если вы не готовы обучать пользователей с учетом объема подлежащего усвоению материала и затрат времени.

Купить инструментальное средство легко, но изменить культуру и организацию работы, чтобы принять это средство и извлечь из него максимальную пользу, гораздо сложнее. Большинство организаций уже привыкли (и чувствуют себя при этом весьма комфортно!) хранить требования в форматах текстовых редакторов или на бумаге. Для перехода к приемам работы с требованиями с использованием ПО необходим новый образ мышления. Использование средств разработки требований требует избавления от старых привычек проведения встреч по выявлению требований. Средство управления требованиями открывает доступ к требованиям в базе данных любому участнику проекта. Некоторые воспринимают это как снижение контроля над требованиями, которым они раньше обладали, над процессом создания требований, или над тем и другим. Другие предпочитают не показывать миру неполные или несовершенные спецификации требований к ПО, но содержание базы данных доступно всем. Скрывая требования до того, как они «готовы», вы упустите возможность показать их многим парам глаз — а это отличный способ выявления возможных проблем.

Люди часто сопротивляются замене вещей, к которым они привыкли, и они обычно чувствуют себя комфортно при работе с требованиями в документах. У них может сложиться впечатление — хотя оно и неправильное, что работать со средством управления требованиями будет сложнее. Надо также не забывать, что большинство пользователей инструментального средства уже и так сильно заняты. Им потребуется выделить дополнительное время на освоение и внедрение средства в свою повседневную работу. В конечном итоге, инструментальное средство не потребует от пользователей больше времени, но прежде им придется изучить его и выработать привычки работы с новым инструментом. Вот несколько рекомендаций, которые позволят разрешить проблемы с принятием нового инструмента пользователями и поспособствуют изменению культуры:

- Назначьте «защитника» инструментального средства, энтузиаста, который сам изучит инструмент вдоль и поперек, будет учить других пользователей и следить, чтобы его использовали как нужно. Это должен быть опытный бизнес-аналитик, который единолично может отвечать за внедрение инструментального средства. Первый энтузиаст инструментального средства будет сотрудничать с другими пользователями в их проектах над внедрением его в повседневную работу, а затем обучать и наставлять других по мере применения этого средства в других проектах.
- Одна из самых больших проблем принятия заключается в том, что пользователи не верят, что от инструментального средства будет польза. Скорое всего это из-за того, что они не понимают сложностей, связанных с ограничениями их существующих ручных процессов. Поделитесь с ними рассказами о том, как отсутствие соответствующего инструментального средства привело к отрицательным результатам, и попросите их вспомнить о примерах из своей практики.
- Несомненно, члены вашей команды умны, но лучше обучить их, чем ждать, когда они сами освоят инструмент. Они, конечно же, додумаются, как осуществлять основные операции, но полный набор возможностей инструмента и их эффективная эксплуатация останутся для них тайной за семью печатями.
- Нельзя ожидать немедленных результатов, поэтому не рассчитывайте, что проект сразу станет успешным из-за первого применения инструментального средства. Прежде наберитесь опыта, работая с инструментом над пробными проектами, и лишь затем применяйте его в важном проекте. Это поможет понять, сколько усилий необходимо для администрирования и поддержки инструментального средства. В главе 31 описывается процесс обучения, связанный с принятием новых средств и приемов.

Распространение и рост популярности инструментальных средств разработки и управления требованиями — заметное направление разработки ПО, которое несомненно продолжится. Вместе с тем слишком много организаций не получают ожидаемой отдачи от вложений в такие средства. Они непра-

вильно оценивают культуру и процессы внутри организации, а также не учитывают усилия по переходу от «документальной» парадигмы требований к подходу на основе инструментальных средств. Приведенные в этой главе рекомендации помогут выбрать и эффективно использовать соответствующие средства. Надо помнить, что средство не может заменить надежного процесса работы с требованиями или членов команды с соответствующими навыками и знаниями.

Что дальше?

- Проанализируйте недостатки вашей существующей организации управления требованиями, чтобы оценить инструментальное средство и понять, оправдает ли оно вложения. Важно, чтобы вы понимали причины существующих проблем; не нужно слепо верить, что инструмент магическим образом разрешит их.
- До сравнения предлагаемых продуктов оцените готовность вашей команды к использованию инструмента. Вспомните предыдущие попытки включить новые инструментальные средства в процесс разработки ПО. Проанализируйте их успех или неудачу, чтобы настроить себя на успех в этот раз.

Часть V

Реализация процесса построения требований

Глава 31

Совершенствование процессов работы с требованиями

Все согласились, что последние несколько проектов прошли не очень гладко. Будучи ведущим бизнес-аналитиком, Джоанн знала, что проблемы с требованиями стали причиной как минимум части проблем. Уровень образования и опыта бизнес-аналитиков в различных проектах сильно различался. Они использовали различные подходы к разработке и управлению требованиями, делая максимум возможного в рамках своего опыта и знаний. Они по-разному организовывали свои требования. В одних командах применялись эффективные процессы изменения требований, что способствовало упорядочению работы над проектами, а в других командах реагировали пассивно и каждый запрос на изменение нарушал работу команды. Уровень разочарования зашкаливал.

Джоанн попыталась заняться обучением своих менее опытных коллег бизнес-аналитиков, но способность к восприятию материала у разных бизнес-аналитиков оказалась разной. Некоторые команды в компании Джоанн хорошо работали над своими требованиями, и число проблем, возникающих в этих проектах, было значительно ниже. Джоанн поняла, что хорошо бы поднять на более высокий уровень производительность работы с требованиями во всех командах. Возможно, именно сейчас наступило подходящее время для серьезной работы над улучшением приемов работы с требованиями. Но согласятся ли подыграть бизнес-аналитики и члены других команд? Действительно ли руководство настроено на снижение напряжения в этих болевых точках? Изменится ли что-то в этот раз или все заглохнет в болоте безразличия, как уже было до этого?

В предыдущих главах описано несколько десятков приемов разработки требований, которые можно применять не практике. Применение рекомендованных приемов — это основа совершенствования процесса разработки ПО. Если кратко, совершенствование технологических процессов заключается в использовании тех методов, которые показывают хорошие результаты, и отказе от тех, которые в прошлом приносили одну головную боль. Однако путь к совершенствованию производительности вымощен фальстартами, противодействием людей, которых это затрагивает, и постоянной боязнью, что не

хватает времени на решение текущих задач, не говоря уже о совершенствовании программ.

Конечная цель совершенствования процессов разработки ПО — уменьшить стоимость создания и сопровождения продукта, что повышает отдачу проектов. Существует несколько способов достижения этой цели:

- исправление проблем, выявленных в предыдущих или текущих проектах, которые возникали из-за недостатков в технологическом процессе;
- предупреждение и предотвращение проблем, которые возможны в будущих проектах;
- применение технологических процессов, более действенных, чем те, что используются в настоящее время.

Если кажется, что методы, используемые вашей командой в настоящем, работают хорошо или сотрудники настаивают на этом, несмотря на свидетельства об обратном, не всегда очевидна необходимость менять подходы к работе. Тем не менее даже успешные организации — разработчики ПО могут испытывать затруднения, когда им приходится заниматься крупными проектами, новыми клиентами, удаленным взаимодействием, когда ужесточаются сроки или возникает необходимость осваивать новые области применения ПО. Методы, отлично зарекомендовавшие себя при работе команды из пяти человек с одним клиентом, не удается масштабировать на 100 человек, распределенных по трем часовым поясам, обслуживающих 50 корпоративных клиентов. По меньшей мере, вам стоит иметь представление о других способах разработки требований, которые могут оказаться ценным дополнением к тем, что вы используете сейчас.

В этой главе рассказано, как требования связаны с процессами проекта и заинтересованными в проекте лицами. Мы познакомим вас с некоторыми основными принципами совершенствования процессов разработки ПО и рекомендуемым циклом совершенствования процессов. В заключение вы узнаете о том, как улучшить «дорожную карту», необходимую для реализации модернизированных процессов разработки требований.

Как требования связаны с другими составляющими проекта

Требования составляют основу любого хорошо организованного проекта по разработке ПО, поддерживая технические и организационные задачи. Изменения способов разработки и управления требованиями повлияют на эти задачи, и наоборот. Рис. 31-1 иллюстрирует некоторые взаимосвязи между требованиями и другими процессами проекта; в последующих разделах кратко описаны взаимосвязи этих процессов.

Рис. 31-1. Взаимоотношения требований и других процессов проекта

Планирование проекта Требования служат основой для процесса планирования проекта. Те, кто отвечает за планирование, выбирают подходящий жизненный цикл разработки ПО и разрабатывают ресурсные сметы и график работы на основе требований. Эта процедура может выявить невозможность реализации всего желаемого набора возможностей за отведенное время при выделенных ресурсах. В результате планирования иногда становится ясно, что следует сузить проект или выбрать инкрементальную модель или модель постадийной реализации необходимой функциональности. В проекте гибкой разработки границы проекта задаются посредством набора пользовательских историй в резерве проекта или выпуска, которые поэтапно реализуются в каждой итерации. Объем работы, запланированный на будущие итерации основан на измерениях скорости в предыдущих итерациях.

Отслеживание и управление проектом Отслеживание подразумевает мониторинг состояния каждого требования, чтобы менеджеру проекта было ясно, действительно ли конструирование и проверка выполняются согласно принятому плану. Если это не так, руководство, клиенты или другие заинтересованные лица могут потребовать изменить границы проекта через процесс управления изменениями. Это может привести к изменению набора требований, над которым ведется работа. В проекте гибкой разработки границы корректируются путем переноса низкоприоритетных задач на будущие итерации, если каждая итерация должна завершаться четко по графику.

Управление изменениями После создания базовой версии набора требований все последующие изменения выполняются только через определенный процесс управления изменениями. Изменения в требованиях приводят к из-

менениям объема остающейся работы и приоритетов элементов этой работы. Отслеживание требований позволяет оценить масштаб влияния этих изменений. Как говорится в главе 28, процесс управления изменениями позволяет гарантировать, что соответствующие люди принимают информированные решения по принятию соответствующих изменений в требованиях.

Приемочное и системное тестирование Пользовательские и функциональные требования представляют собой важнейшие входные данные соответственно для приемочного и системного тестирования системы. Если предполагаемое поведение ПО в различных условиях не определено четко, тестировщикам будет чрезвычайно трудно обнаружить дефекты и проверить, вся ли запланированная функциональность реализована должным образом. Одна моя знакомая недавно рассказала мне такую историю: «Мне поручили написать план тестирования по спецификации требований к ПО, созданной другим аналитиком. Я потратила существенно больше запланированного времени, потому что было очень трудно разобраться в описании функциональности. Связанные функции иногда находились в совершенно неожиданных местах спецификации. А в некоторых случаях автор спецификации приводил пространные описания параметров, которые *не были* выбраны для реализации, и только после этого переходил к тем, которые решено реализовывать. Это было очень неприятно».

Процесс разработки Требования предоставляют основу для работы по дизайну и реализации и связывают воедино различные рабочие продукты. Проверяйте построенные компоненты, чтобы удостовериться, что каждый из них точно соответствует требованиям. Модульное тестирование позволяет проверить, удовлетворяет ли код модулей спецификациям и соответствующим требованиям. Отслеживание требований позволяет задокументировать, какие элементы конструкции и кода ПО были получены на основе того или иного требования.

Пользовательская документация Однажды я делил офис с техническими писателями, которые готовили пользовательскую документацию для сложных программных продуктов. Я спросил одного из них, почему они так часто задерживаются на работе. Он ответил: «Мы находимся в конце пищевой цепочки, — ответил он. — Нам приходится очень быстро реагировать на изменения в пользовательском интерфейсе и функциях, которые убирают или добавляют в последний момент». Требования к продукту содержат данные для процесса разработки пользовательской документации, так что плохо сформулированные или запаздывающие требования порождают проблемы в документации. Не удивительно, что страдальцы в конце цепочки работы с требованиями — технические писатели и тестировщики — часто с энтузиазмом поддерживают совершенствование приемов разработки требований и их привлечение на ранних стадиях процесса.

Требования и различные группы заинтересованных лиц

На рис. 22-2 показаны некоторые заинтересованные в проекте лица, взаимодействующие с группой разработки ПО, и вклад, который они вносят в процесс создания требований проекта. Если вы бизнес-аналитик или менеджер проекта, объясните заинтересованным лицам в каждой области, какую информацию и другое содействие, необходимое для успеха проекта, вы хотите получить от них. Согласуйте форму и содержание взаимодействия между разработчиками и теми, кто занимается другими вопросами, связанными с продуктом, — спецификацией системных требований, документацией рыночных требований или набором пользовательских историй.

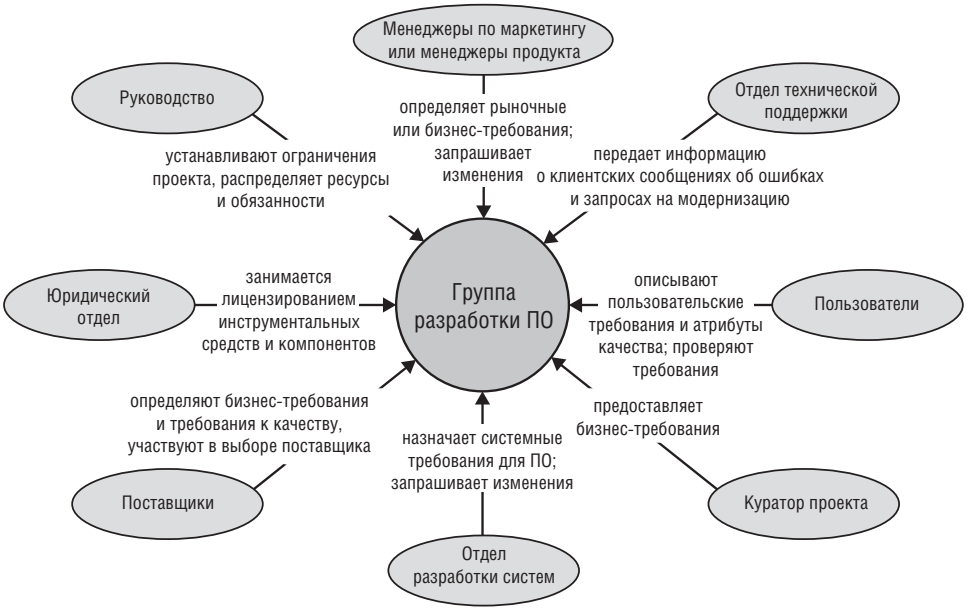


Рис. 31-2. Связанное с требованиями взаимодействие между разработчиками ПО и другими заинтересованными лицами проекта

С другой стороны, бизнес-аналитик и менеджер проекта должны поинтересоваться у других заинтересованных лиц, что они хотят получить от группы разработчиков для облегчения своей работы. Какая информация о технической осуществимости поможет отделу маркетинга лучше спланировать концепции продуктов? Какие отчеты по состоянию требований дадут менеджерам ясное представление о ходе проекта? Какое взаимодействие с отделом разработки систем позволит верно судить о том, что системные требования правильно распределены между программной и аппаратной подсистемами? Бизнес-аналитик и менеджер проекта должен стремиться строить взаимоотношения сотрудничества между разработчиками и другими заинтересованными в проекте сторонами.

Как добиться готовности к изменениям

Когда группа разработчиков ПО изменяет свои процессы работы с требованиями, их взаимодействие с другими заинтересованными в проекте сторонами также изменяется. Люди не любят, когда их принуждают выходить за пределы их зоны комфорта, поэтому будьте готовы к некоторому сопротивлению вашим начинаниям. Узнайте причины сопротивления, чтобы иметь возможность как понимать, так и гасить его.

Часто сопротивление вызвано страхом перед неизвестным. Чтобы уменьшить боязнь, сообщите логическое обоснование улучшений. Объясните преимущества, которые другие группы получат от нововведений. Вначале озвучьте свою точку зрения: «Вот проблемы, которые все мы испытали. Какие проблемы вы видите со своей точки зрения? Можем ли мы вместе найти лучший способ решения вопросов?» Привлечение других заинтересованных лиц в мероприятия по улучшению делают их соавторами решений.

Далее перечислены некоторые формы сопротивления, с которыми вы можете столкнуться:

- Люди, которые уже слишком заняты в основном проекте, не считают, что могут тратить свое время на освоение других приемов. Но если не потратить это время, не стоит ожидать, что следующий проект пройдет более гладко, чем текущий.
- Процесс управления изменениями можно рассматривать как барьер, воздвигаемый разработчиками, чтобы затруднить изменения. В действительности процесс управления изменениями — это структура, а не барьер. Она позволяет хорошо информированным людям принимать отличные бизнес-решения и доводить их до других участников. Команда разработчиков ПО отвечает за то, чтобы процесс внесения изменений действительно работал. Если новые процессы не принесут лучших результатов, люди найдут способы их обойти.
- Некоторые разработчики и менеджеры считают документирование и проверку требований пустой бюрократией, не дающей им заниматься «настоящей» работой — написанием кода. Если вы сможете объяснить, насколько дорого обходится переписывание кода, пока команда пытается понять, что должна делать система, разработчики и менеджеры будут больше ценить необходимость хороших требований. Пропущенные требования могут снизить прибыльность в процессе использования программного продукта, потому что постоянно приходится тратиться на производство обновлений.

Когда людей просят изменить методы работы, их естественная реакция: «А что мне с этого будет?» Однако изменения процессов не всегда приводят к поразительным и немедленным результатам для всех, кого процесс затрагивает. Более важный вопрос, кстати, на который любой, кто отвечает за совершенствование процесса, должен уметь отвечать убедительно: «Что *нам* с этого будет?» Любое изменение процессов должно предлагать перспекти-

ву ясных преимуществ команде, работающей над проектом, организации-разработчику, компании и/или клиентам Заинтересованные лица, которых просят больше времени потратить на помощь в совершенствовании требований могут посчитать это как просто дополнительную работу для них. Но представьте, что они поймут, что эти затраты обернуться значительным выигрышем в виде уменьшения переделок на более поздних этапах проекта, сокращения затрат на поддержку и увеличения ценности продукта для пользователей. Такое понимание может сделать их более сговорчивыми и готовыми потратить свое время прямо сейчас.

Это естественно, что некоторые заинтересованные лица проекта не в курсе обусловленного требованиями влияния на текущие процессы работы в организации. Поэтому важным методом добиться приверженности к изменению процессов является конструктивное и непредвзятое освещение проблем. Представьте, что разработчики работают над приложением, которое требует значительной поддержки со стороны клиента из-за проблем с пользовательским интерфейсом. Если этими проблемами занимается исключительно отдельное подразделение поддержки, то в команде разработчиков об этих проблемах ничего не знают. Или, допустим, руководство отдало разработку на сторону в попытке сэкономить деньги и время, но не озаботилось разрешением возникших коммуникационных и культурных барьерах. Если менеджеры не в курсе последствий, у них не будет причин менять свой подход для устранения подобных недостатков.

1. Требует, чтобы требования в проекте документировались в надлежащей форме.
2. Работает с бизнес-аналитиком для обеспечения бизнес-требований для каждого проекта.
3. Ожидает, что требования рецензируются соответствующими заинтересованными лицами, включая само руководство, если это нужно.
4. Требует от заинтересованных лиц согласовать требования до реализации каждой части решения.
5. Проверяет, что проектные планы включают время и ресурсы для работы над требованиями.
6. Сотрудничает с другими ключевыми заинтересованными лицами, привлекая их к работе над требованиями.
7. Устанавливает эффективные механизмы и политики работы с изменениями требований.
8. Выделяет средства на приобретение обучающих материалов, книг, инструментов и других ресурсов для вовлеченных в работу над требованиями.
9. Выделяет средства и ресурсы на улучшение процессов работы над требованиями в организации.
10. Выделяет членам команды время, которое они должны потратить на улучшение процессов работы над требованиями.

Рис. 31-3. Действия руководства, служащие признаком приверженности руководства совершенствованию процессов работы над требованиями

Мы часто слышим, как бизнес-аналитики и другие участники говорят, что они не могут изменить какой-то процесс в своей организации без «поддержки руководства». Но слишком часто поддержка руководства выражается всего лишь в разрешении делать что-то иначе. Но будучи квалифицированным специалистом, вы не нуждаетесь в разрешении руководства работать наилучшим известным вам способом — это просто ваша работа. Но вам определенно

требуется *поддержка и готовность* руководства к успешным и устойчивым улучшениям в рамках проекта или всей организации. Без поддержки руководства начинание поддержат только те специалисты, которые уверены в необходимости более качественных требований. Не будет никакого толка, если руководство поначалу заявит свою «поддержку» улучшениям, но при первых же трудностях быстренько вернется к старым процессам. Приверженность качеству подтверждают действия, а не декларации. На рис. 31-3 показаны 10 признаков того, что руководство организации действительно озабочено тем, чтобы иметь отличные процессы работы с требованиями.

Основы совершенствования процессов разработки ПО

Вы читаете эту главу скорее всего потому, что намереваетесь изменить некоторые методы, используемые в настоящее время вашей организацией для конструирования требований. Начиная свой поход за превосходными требованиями, помните о следующих четырех принципах совершенствования процессов работы над ПО (Wiegers, 1996).

1. **Совершенствование технологических процессов должно быть эволюционным и непрерывным** Вместо того чтобы ставить целью абсолютное совершенствование, разработайте несколько модернизированных шаблонов и процедур и начните их реализацию. Корректируйте свои методы по мере того, как ваша команда будет набираться опыта в применении новых методик. Иногда простые и легкие изменения могут привести к существенной выгоде, так что ищите плоды, висящие низко, — проблемные области, с необходимостью улучшений которых согласны все. Подробнее об эффективных приемах, которые можно внедрить, см. табл. 3-2 в главе 3.
2. **Люди и организации изменяются только при наличии стимула** Сильнейший из них — это боль, или трудности в работе. Я не имею в виду искусственно вызываемые трудности в работе вроде сокращения сроков выпуска по инициативе руководства с целью заставить разработчиков работать усерднее, но реальные трудности, которые вы испытывали в предыдущих проектах. Вот какие проблемы могут стать значительным мотивом для изменения процессов работы с требованиями:
   - проект не уложился в график, потому что требования оказались более обширными, чем ожидалось;
   - разработчикам пришлось много работать в неурочное время из-за неправильно понятых или неоднозначных требований;
   - усилия по тестированию потрачены впустую, потому что тестировщики не понимали, что продукт должен делать;
   - нужная функциональность была реализована, но пользователи остались неудовлетворенными вялой производительностью, неудобством работы или другими недостатками качества продукта;

- организации пришлось нести высокие расходы на сопровождение, потому что клиентам потребовалась масса дополнительных функций, которые должны были быть определены во время составления требований;
- в процессе проекта изменения требований были реализованы надлежащим образом, поэтому полученное в результате решение не удовлетворило потребностей клиента;
- корректировки требований были потеряны или перезаписаны из-за того, что несколько бизнес-аналитиков работали над ними одновременно без использования процесса управления версиями;
- клиенты были недоступны, когда нужно было уточнить или выяснить детали требований;
- проблемы, связанные с требованиями, не разрешались вовремя, что приводило к необходимости переделок.

3. **Изменения технологических процессов должны ориентироваться на определенную цель** Прежде чем улучшать процессы, убедитесь, что знаете, куда идете (Potter и Sakry, 2002). Хотите ли вы уменьшить объем работы, которую приходится переделывать из-за проблем с требованиями? Или, может, нужно, чтобы меньше требований оставались незамеченными или чтобы ненужные требования удалялись как можно раньше? Дорожная карта, определяющая пути к бизнес-целям, значительно увеличит ваши шансы на успех в совершенствовании процессов.
4. **Воспринимайте действия по совершенствованию процессов как мини-проекты** Многие попытки внесения усовершенствований терпят неудачу из-за плохого планирования или отсутствия ожидаемых ресурсов. Учтите ресурсы, необходимые для совершенствования процессов, в общих планах по проекту. Выполняйте те же процедуры планирования, учета, нормирования и отчетности, что и для любого другого проекта, с учетом малых масштабов проекта по внесению усовершенствований. Создайте простой план действий для каждой области совершенствования процессов, которую нужно охватить.

> **Внимание!** Наибольшая угроза программе совершенствования процессов разработки ПО — недостаточно серьезное отношение руководства к своим обязательствам, что влечет за собой череду реорганизаций, перетасовывающих участников и приоритеты программы.

Все члены команды имеют возможность — и обязанность — активно улучшать свой рабочий процесс. Если вы самостоятельно решите какую-то проблему, ваши коллеги по команде скорее всего заметят преимущество и без промедления примут новый процесс выполнения задачи. Тем не менее масштабная инициатива по совершенствованию технологических процессов может преуспеть, только если руководство готово выделять ресурсы, определять ожидания и устанавливать ответственность членов команды за их вклад в программу по совершенствованию.

Шутливые советы по совершенствованию технологических процессов

У опытного руководителя программ совершенствования технологических процессов со временем накапливается перечень коротких и содержательных наблюдений об этой трудной области. Вот некоторые из тех, что я собрал на протяжении ряда лет:

- не откусывайте кусок, который не сможете прожевать (если вы попробуете глобально изменить работу, велика вероятность, что ваша команда просто «подавится»);
- получайте максимум удовольствия от малых побед (больших побед у вас будет немного);
- давите мягко, но постоянно (ведите команду к лучшему будущему, поощряя инициативу и постоянно реализуя ее маленькими шажками);
- концентрация, концентрация, концентрация (занятая команда разработчиков ПО может выполнять лишь три, или две, или, может быть, всего одну инициативу по совершенствованию за раз. Но никогда не беритесь меньше, чем за одну инициативу);
- ищите союзников (в каждой команде есть свои энтузиасты, которые с готовностью испробуют новые шаблоны и процедуры и поделятся полученным опытом с руководителем программы; растите их; благодарите их; награждайте их);
- планы, не реализуемые в действиях, нельзя считать полезными (легко оценить процесс и написать план действий; трудно заставить людей работать по-новому в ожидании лучших результатов, но это — единственный полезный результат процесса совершенствования);
- участвовать должны все (перетяните на свою сторону членов команды, которые должны реализовывать изменения, привлекая их в процессы оценки и обнаружения действий по улучшению процессов).

Анализ первопричин

Очень важно сосредоточить свое ограниченное время и бюджет на улучшение процессов там, где оно больше всего требуется. Если вы в состоянии определить причины любых недостатков процесса, с которым вы столкнулись, можно определить их как цели с высоким потенциалом для улучшений.

Задача *анализа первопричин* (root cause analysis) заключается в определении базовых факторов, вносящих свой вклад в наблюдаемую проблему, с отделением симптомов от их причин. Анализ первопричин требует последовательной и многократной постановки вопроса «почему» существует проблема, при этом каждый раз испытывается причина, указанная в ответе на предыдущий вопрос «почему». Выполняйте анализ первопричин до внедрения изменений в процесс, чтобы определить, почему существующие подходы не дают требуемых результатов. В противном случае легко оказаться в ситуа-

ции слепого котенка: пробовать новые методы, не имея никакой уверенности, что они позволят решить реальные проблемы.

Иногда неясно, что представляет собой проблему и что первопричину. Некоторые симптомы и первопричины связываются в крепкий клубок, в котором один симптом становится первопричиной другого. Представьте, что симптом заключается в том, что на этапе выявления слишком много требований остаются незамеченными. Одной из возможных первопричин может быть то, что бизнес-аналитик не задал нужных вопросов. Эта первопричина является симптомом другой проблемы: люди с ролью бизнес-аналитика не знают, как это делать качественно.

Диаграмма причин и следствий (cause-and-effect diagram) — ее еще называют *диаграммой «рыбьей кости»* (fishbone diagram) или *диаграммой Ишикавы* (Ishikawa diagram) по имени его изобретателя Каору Ишикавы — это полезный способ представления результатов анализ первопричин. На рис. 31-4 показана диаграмма причин и следствий, которая частично анализирует проблему, заключающуюся в том, что проектные команды организации неоднократно не справляются с завершением проектов в срок. На этой диаграмме «ребра», отходящие от «позвоночника», показывают ответы на вопрос «Почему команды не завершают проекты вовремя?» Дополнительные ребра показывают ответы на последующие вопросы «почему». В конечном итоге анализ вскрывает фундаментальные первопричины на верхних ветках-«ребрах».

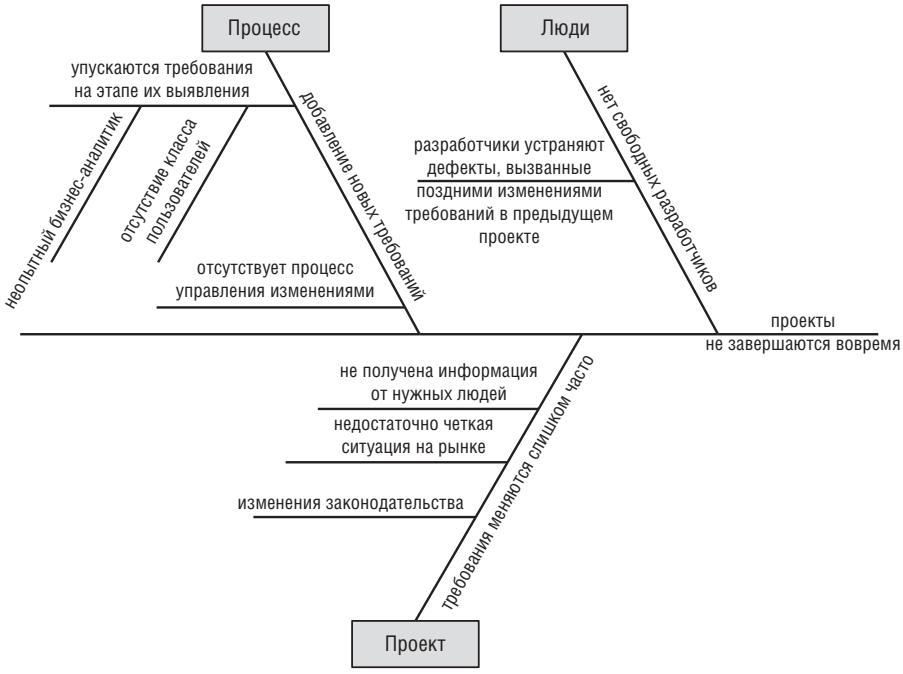


Рис. 31-4. Диаграмма причин и следствий, определяющая первопричины для выявленных симптомов проблемы

При использовании такого анализа не нужно разбираться со всеми первопричинами. Принцип Парето, известный также как правило «80/20», говорит, что примерно 20% важных первопричин вызывают примерно 80% наблюдаемых проблем. Даже простой анализ первопричин может выявить такие важные причины, на разрешение которых нужно направлять действия по совершенствованию требований.

Цикл совершенствования технологических процессов

На рис. 31-5 изображен цикл совершенствования технологических процессов, в эффективности которого я убедился. Этот цикл показывает, насколько важно знать собственное местоположение, прежде чем отправляться куда-либо еще, прокладывать курс и оценивать полученные уроки как часть непрерывного процесса совершенствования процессов.

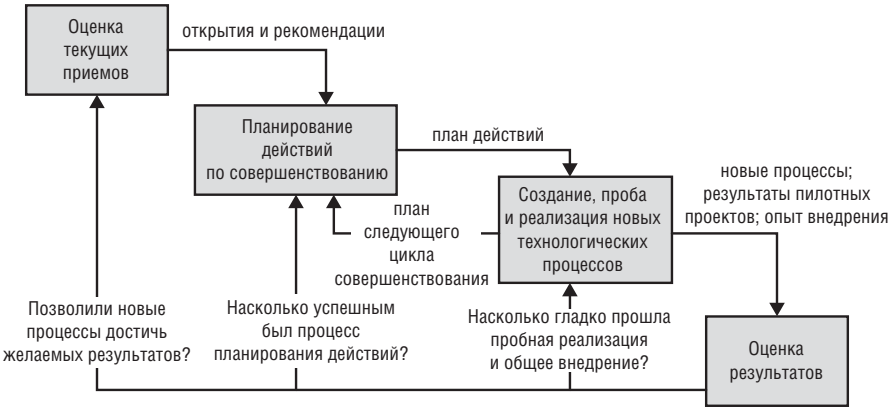


Рис. 31-5. Цикл совершенствования процессов при разработке ПО

Оценка текущих приемов

Первое, с чего следует начать любое совершенствование, — это оценка текущих технологических процессов, используемых организацией, и определение их сильных и слабых сторон. Оценка дает фундамент для выбора желаемых изменений. Кроме того, вы делаете видимыми реальные рабочие процессы в организации — часто они отличаются от официально утвержденных или описанных в документации. И вы обнаружите, что разные члены команды имеют весьма разные взгляды на то, какие процессы используются сейчас.

Текущие процессы работы над требованиями можно оценивать разными способами. Если вы воспользовались хотя бы одной из задач в разделе «Что дальше?» предыдущих глав, вы уже начали неформальную оценку своих приемов работы над требованиями и их результатов. В Приложении Б при-

водятся десятки симптомов стандартных проблем с требованиями вместе с возможными причинами и решениями. Структурированные анкеты предлагают более систематический подход, который позволит без особых затрат выявить важную информацию. Интервью и обсуждения с членами команды дают более точное и полное понимание, чем анкетирование. К результатам формальной оценки относится список обнаруженных сильных и слабых сторон в текущих процессах, а также, рекомендации по совершенствованию.

Вы можете использовать анкету в Приложении А для идентификации текущих приемов конструирования требований в вашей организации. Этот тест помогает понять, какие из ваших процессов работы с требованиями более всего нуждаются в совершенствовании. Но если вы поставили низкую оценку какому-либо вопросу, это не стоит считать достаточной причиной, чтобы приступать к его решению немедленно, а может, и вообще. Направьте усилия на совершенствование приемов, вызывающих наибольшие трудности и подвергающих риску успех ваших будущих проектов.

Планирование действий по совершенствованию

В рамках подхода, в котором действия по совершенствованию процессов считаются проектами, после оценки приемов напишите план действий (Potter и Sakry, 2002). Тактический план действий нацелен на конкретные области совершенствования, например на процесс сбора требований или процедуру назначения приоритетов. В каждом плане действий должны быть указаны цели действий по совершенствованию, участники и конкретные задачи. Без плана можно легко пропустить важные задачи. План также дает возможность отслеживать выполнение процесса совершенствования, отмечая выполнение отдельных задач.

На рис. 31-6 показан шаблон плана совершенствования технологических процессов, которым я многократно пользовался. Включайте в каждый план действий не более 10 пунктов и ограничьте его двумя-тремя месяцами. Например, я видел план совершенствования управления требованиями, включавший в себя следующие задачи:

1. составить проект процедуры управления изменениями;
2. проверить и модифицировать процедуру управления изменениями;
3. провести пробное испытание процедуры управления изменениями на примере проекта А;
4. пересмотреть процедуру управления изменениями на основе обратной реакции по пробному испытанию;
5. оценить инструментальные средства отслеживания проблем и выбрать одно из них для поддержки процедуры управления изменениями;
6. приобрести выбранное инструментальное средство выявления проблем и настроить его для поддержки конкретного процесса;
7. внедрить новую процедуру управления изменениями и инструментальное средство в организации.

Поручите каждую задачу конкретному человеку, который будет отвечать за ее выполнение. Не назначайте «всю команду» ответственной за задачи. Работу выполняют не команды, а отдельные люди.

Если вам нужно выполнить более 10 задач, в первый цикл включите самые важные вопросы, а остальные отложите до следующей версии плана действий. Помните, изменения всегда должны быть инкрементальными и непрерывными. Дорожная карта совершенствования технологических процессов, описанная далее в этой главе, показывает, как группировать множество задач совершенствования отдельных действий для общего плана совершенствования процессов разработки ПО.

План совершенствования процессов работы с требованиями

Проект: _____ **Дата:** _____

<название проекта> *<дата написания плана>*

Цели:

<Сформулируйте несколько целей, которых вы желаете достичь в результате успешного выполнения этого плана. Формулируйте цели с точки зрения бизнеса, а не изменений в процессах.>

Мера успеха:

<Опишите, как вы будете определять, есть ли желаемые результаты в проекте вследствие изменений в процессах.>

Масштаб влияния на организацию:

<Опишите широту влияния изменений на процессы, описанные в этом плане.>

Сотрудники и участники:

<Определите, кто будет претворять в действие этот план, их роли и время, которое они должны затратить (в часах в неделю либо в процентах).>

Процесс учета и отчетности:

<Опишите, как будет учитываться продвижение в выполнении задач этого плана, и кто будет получать отчеты по состоянию, результатам и проблемам.>

Зависимость, риски и ограничения:

<Определите, какие внешние факторы нужны для успешной реализации этого плана или воспрепятствовать ей.>

Ожидаемые сроки выполнения всех задач:

<Когда вы ожидаете полного выполнения этого плана?>

Задачи:

<Запишите от 3 до 10 задач для каждого плана действий.>

Задача	Ответственное лицо	Срок выполнения	Описание действий	Результаты	Необходимые ресурсы
<Номер по порядку>	<Человек, несущий ответственность за выполнение>	<Ожидаемая дата выполнения>	<Действия, которые следует выполнить для реализации конкретной задачи>	<Процедуры, шаблоны или другие ресурсы процесса, которые будут созданы>	<Любые необходимые внешние ресурсы, в том числе материалы, инструменты, документы или люди>

Рис. 31-6. Шаблон плана совершенствования процессов разработки ПО

Создание, проба и реализация новых технологических процессов

До сих пор вы оценивали свои текущие приемы работы с требованиями и составляли план преобразования областей, изменения в которых, по вашему мнению, дадут наилучшие результаты. Пришло время для самого трудного: реализации плана.

Реализация плана действия означает разработку процессов, которые, по вашему мнению, дадут лучшие результаты, чем те, что применяются сейчас. Не ожидайте с первой попытки получить совершенные процессы. Многие приемы, кажущиеся неплохими в теории, оказываются менее эффективными на практике. Поэтому предусмотрите пилотный проект для небольшого числа новых процедур или шаблонов документов, которые вы создаете. Опыт пилотного выпуска пригодится для настройки нового процесса. Это увеличивает вероятность того, что он покажет хорошие результаты и будет хорошо воспринят при внедрении. Составляя пилотные проекты, учитывайте следующие советы:

- выбирайте для участия в пилотных проектах людей, которые будут относиться к новым приемам беспристрастно и смогут дать им оценку. Это могут быть как сторонники, так и скептики, но они не должны быть ярыми противниками предлагаемых приемов;
- чтобы результаты было легко истолковать, определите количество критериев, по которым команда будет оценивать пилотный проект;
- определите заинтересованных лиц, которых следует проинформировать о том, что представляет собой пробный проект и почему он выполняется;
- подумайте о возможности испытания новых процессов по частям в разных пробных проектах. Так вам удастся вовлечь в испытание больше людей, что увеличивает осведомленность, количество откликов и сторонников;
- для более полной оценки поинтересуйтесь у участников пилотных проектов, что бы они почувствовали, если бы им пришлось вернуться к старым приемам работы.

В случае успеха пилотного проекта вы готовы сделать все последние коррективы в процесс и представить их участникам для реализации. Даже у мотивированных и восприимчивых команд есть ограниченная возможность восприятия изменений, так что не слишком много ожидайте от первого проекта. Напишите план внедрения, в котором укажите, как вы будете распределять новые методы и материалы в команде, работающей над проектом, обучать их и помогать им. Также, продумайте, как руководство будет сообщать свои ожидания по поводу новых процессов.

Оценка результатов

Последний шаг цикла совершенствования процесса — это оценка предпринятых действий и достигнутых результатов. Она поможет команде добиться еще

больших успехов в реализации последующего совершенствования. Оцените, насколько гладко прошли пилотные проекты и как эффективно они разрешили неопределенности в отношении новых процессов. Собираетесь ли вы менять что-либо в следующих пилотных проектах?

Как прошло общее внедрение новых технологических процессов? Удалось ли вам довести до сведения каждого информацию о пользе новых процессов или шаблонов? Смогли ли участники понять и эффективно применить новые процессы? Собираетесь ли вы менять что-либо при проведении следующего внедрения?

Критически важно оценить, приводят ли новые процессы к желаемым результатам. Польза некоторых новых технических и управленческих приемов проявляется быстро, выгода применения других видна не сразу. Например, вы должны быстро получить возможность оценить, облегчает ли новый процесс редактирование требований. В то же время ценность нового шаблона спецификации требований к ПО станет ясна со временем, когда аналитики и клиенты привыкнут к порядку документирования требований в определенном формате. Дайте новым приемам поработать достаточное время и выберите мерило оценки пользы каждого изменения процесса.

Смиритесь с необходимостью учета периода обучения: производительность падает, пока люди приспосабливаются к новым способам работы, как показано на рис. 31-7. Кратковременное падение производительности, иногда называемое «долиной отчаяния» — это часть необходимого вклада, который ваша организация вносит в совершенствование процессов. Если люди не понимают этого, велико искушение отказаться от работы по совершенствованию процесса до того, как она начнет приносить плоды; прибыль от таких инвестиций равна нулю. Расскажите своим менеджерам и сотрудникам о периоде обучения и следите за инициативой на протяжении всего проекта.

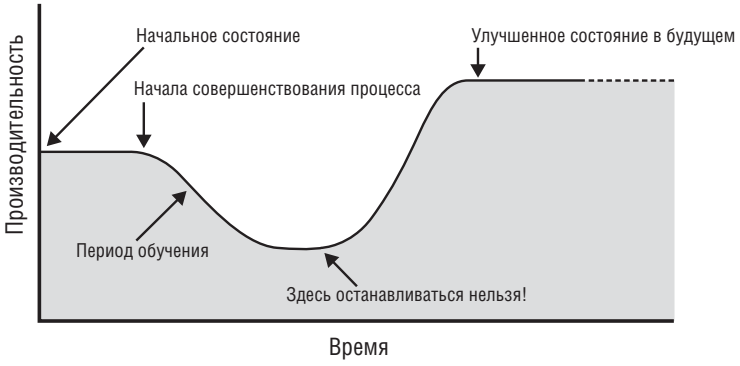


Рис. 31-7. Период обучения — неизбежная составляющая совершенствования технологических процессов

Документы процесса разработки и управления требованиями

Высокопроизводительные проекты отличаются эффективными процессами на всех этапах создания требований: выявления, анализа, спецификации, проверки и управления. Для облегчения выполнения этих процессов каждой организации необходим набор *документов процесса* (process assets) (Wiegers, 1998b). Любой процесс определяют выполняемые действия и получаемые результаты; документы процесса помогают команде выполнять процессы последовательно и эффективно. Эти документы позволяют участникам проекта понять, какие шаги им следует предпринять и каких результатов от них ждут. Набор документов процесса включает в себя документы различных типов, которые описаны в табл. 31-1.

Табл. 31-1. Типы документов процесса

Тип	Описание
Контрольный список	Перечень действий, продуктов или других элементов, которые следует записать или проверить. Это будильники для памяти. Он помогает занятым людям не упускать из виду важных деталей
Пример	Пример рабочего продукта конкретного типа. Накапливайте хорошие примеры по мере того, как ваша команда их создает
План	Схематичное описание цели и всего необходимого для ее достижения
Политика	Руководящий принцип, задающий ожидания относительно поведения, действий и конечной версии продукта. Процессы должны удовлетворять политикам
Процедура	Пошаговое описание последовательности задач, ведущей к выполнению действия. Опишите, какие задачи следует выполнить, и определите роли их исполнителей. Не включайте в процедуру информацию по обучению. Основные документы могут содержать учебную информацию и подсказки для процесса или процедуры
Описание процесса	Задокументированное определение набора действий, исполняемого с какой-либо целью. Описание процесса может включать цель процесса, ключевые вехи, участников, этапы взаимодействия, входные и выходные данные, связанные с процессом артефакты и способы настройки процесса для различных ситуаций
Шаблон	Образец, используемый в качестве руководства по созданию завершенного продукта. Шаблоны ключевых документов проекта содержат много разделов для сбора и организации информации. Указания, которые содержит шаблон, помогут автору документа использовать его эффективно. Другие шаблоны определяют структуру, полезную для написания определенного типа информации, такой как функциональное требование, атрибут качества, бизнес-правило или пользовательская история

На рис. 22-6 определены некоторые ценные документы процесса для создания требований. Эти элементы не должны быть длиннее, чем это необходимо, чтобы позволить членам команды выполнять их последовательно и эффективно. Для них не нужен отдельный документ; общий процесс управления требованиями может включать процедуру управления изменениями, процедуру проверки состояния и контрольный список анализа последствий. Храните эти элементы в общей библиотеке документов процесса, чтобы гарантировать простоту доступа и постоянную доступность, а также обеспечьте механизмы их совершенствования с накоплением опыта (Wiegers, 1998b). Многие из документов, перечисленных на рис. 31-8, можно загрузить с вебсайта книги.

Документы процесса для разработки требований	Документы процесса управления требованиями
• Процесс разработки требований • Процедура назначения требований • Процедура назначения приоритетов требованиям • Документ о концепции и границах • Шаблон вариантов использования • Шаблон спецификации требований к ПО • Контрольный список рецензирования требований	• Процесс управления требованиями • Процедура отслеживания состояния требований • Процесс управления изменениями • Шаблон устава совета по управлению изменениями • Контрольный список анализа влияния, оказываемого изменениями • Процедура отслеживания требований

Рис. 31-8. Основные документы процесса для разработки и управления требованиями

Далее кратко описан каждый из документов, перечисленных на рис. 31-8, а также указаны ссылки на главы, где они объясняются в деталях. В каждом проекте следует планировать работу с требованиями на основе принятых в организации процессов. Например, в крупном проекте с большим числом классов пользователей и других заинтересованных лиц в разных местах будет польза от письменного плана выявления требований, в котором указаны приемы, применяемые для выявления требований, а также кто, когда и где должен выполнять эту работу. В проекте, в котором заинтересованные лица находятся в одном месте и тесно сотрудничают друг с другом, можно обойтись более простым и близким к гибкой разработке процессом.

Документы процесса разработки требований

Приведенные здесь рекомендации позволят командам лучше выполнять работу по выявлению, анализу, спецификации и утверждению требований в своих проектах.

Процесс разработки требований описывает, как определить и классифицировать заинтересованных в проекте лиц в предметной области, а также как планировать действия по выявлению требований. Кроме того, здесь перечислены различные документы, касающиеся требований, модели, которые, как ожидается, будут созданы при выполнении проекта. Процесс разработки

требований также определяет особенности анализа и проверки требований. Подробнее о содержимом плана выявления требований см. главу 7.

Процедура назначения требований описывает, как назначать высокоуровневые требованиям к продукту конкретным подсистемам при разработке систем, состоящих из программных и аппаратных компонентов или множественных программных подсистем. Подробнее о назначении требований см. главу 26.

Процедура назначения приоритетов требований описывает приемы и инструменты, используемые для определения приоритетов требований и динамической корректировки содержимого резерва в проекте. Подробнее о назначении приоритетов см. главу 16.

Шаблон документа о концепции и границах проекта помогает куратору проекта и бизнес-аналитику осмысливать бизнес-цели, метрики успех, концепцию продукта и другие элементы бизнес-требований. В главе 5 показан шаблон для этого документа.

Шаблон вариантов использования задает стандартный формат для описания задач, которые пользователям необходимо выполнять с помощью программы. Подробнее об этом см. главу 8.

Шаблон спецификации требований к ПО представляет собой структурированный, последовательный способ организации функциональных и нефункциональных требований. Подумайте о возможности применения более чем одного шаблона, чтобы учесть различные типы и масштабы проектов, выполняемые вашей организацией. В главе 10 описан пример шаблона спецификации требований к ПО.

Контрольный список рецензирования требований Официальное рецензирование документов, содержащих требования, — мощное средство повышения качества ПО. В списке рецензирования описано много типичных ошибок, присутствующих в документах требований, что позволяет рецензенту сосредоточиться на стандартных проблемных областях. В главе 17 содержатся примеры контрольных списков рецензирования требований.

Документы процесса управления требованиями

Следующие элементы помогают командам управлять наборами задокументированных требований.

Процесс управления требованиями описывает действия работающей над проектом команды для различения версий требований, определения базовых версий, работой с изменениями требований, различными версиями документации по требованиям, учетом и отчетностью о состоянии требований и накоплением информации по отслеживанию (см. главу 27). Пример описания процесса управления требованиями вы можете найти в книге «CMM Implementation Guide» (Caputo, 1998).

Процедура отслеживания состояния требований подразумевает мониторинг состояния каждого функционального требования и отчетность по нему.

Больше об учете статуса требований вы можете прочитать в главе 18.

Процесс управления изменениями определяет пути предложения, передачи, оценки и разрешения нового требования или его модификации. В главе 28 детально описан процесс управления изменениями.

Шаблон устава совета по управлению изменениями Как говорится в главе 28, устав совета по управлению изменениями описывает состав, функции и рабочие процедуры этого совета.

Контрольный список анализа последствий изменений в требованиях Как показано в главе 28, анализ последствий помогает вам рассмотреть задачи, побочные эффекты и риски, связанные с реализацией каждого изменения в требованиях, а также оценить объем работы по задачам.

Процедура отслеживаемости требований определяет, кто предоставляет данные по отслеживаемости, которые позволяют отслеживать связи требований с другими артефактами проекта, кто их собирает и управляет ими и где они хранятся. Подробнее об отслеживании связей требований см. главу 29.

Достигнута ли цель?

Как и у всех путешествий, у проекта улучшения процессов должна быть цель. Если не определить конкретных целей по улучшению, люди не смогут работать согласованней, а вы не сможете сказать, есть ли движение вперед, не сможете определять приоритеты задач и сказать, когда цель достигнута. *Метрика* — измеримая характеристика проекта, продукта или процесса. *Ключевые показатели производительности* (KPI) — это метрики, привязанные цели и служащие мерилом продвижения проекта к достижению определенной цели или результата. Набор KPI-показателей может отображаться на контрольной панели, показывая приближение к целям.

При определении целей по совершенствованию процессов нужно иметь в виду два обстоятельства. Во-первых, надо помнить, что совершенствование процесса ради самого совершенствования бессмысленно. Поэтому спросите себя, действительно ли достижение цели даст искомый рост бизнес-ценности. Во-вторых, не стоит разочаровывать членов команды, ставя цели, которые нереально достичь, поэтому нужно хорошенько подумать, достижима ли поставленная цель в вашей среде. Чтобы цель улучшения была разумной, ответ на оба вопроса должен быть положительным.

Можно измерять самые разные аспекты работы над требованиями в проекте, в том числе размер проекта, качество требований, состояние требований, деятельность по внесению изменений, а также усилия, потраченные на разработку и управление требованиями (Wiegers, 2006). Кроме того, измерение достижения проектом поставленных бизнес-целей будет отражать, насколько правильно работа над требованиями была направлена на цель. Но в проекте совершенствования процессов нужно выбрать цели измерения, по которым можно судить, оправдывают ли себя затраты на улучшение настолько, насколько вы ожидали. Ранее в этой главе мы говорили, что совершен-

ствование процесса должно основываться на целях, а самым лучшим мотивирующим к изменению фактором являются трудности или неприятности, которые испытала организация в предыдущих проектах. Поэтому KPI нужно выбирать, определяя количественные цели улучшения, после чего можно определить, уходят ли трудности, заставившие поставить такие цели.

Имейте в виду, что нельзя количественно измерять прогресс, если не установить базовый уровень, точку отсчета, определяющую состояние дел на сегодняшний день. В идеале, надо измерить текущее значение какого-то индикатора, после чего задать желаемое целевое значение, которого нужно достичь через определенное время, и направить действия по совершенствованию процессов на получение этого результата. В реальности, во многих организациях, занимающихся разработкой ПО, отсутствует культура измерения, поэтому у них будут трудности с определением таких количественных базовых уровней. Тем не менее сложно сказать, насколько вы приблизились к цели, если у вас нет ни исходной точки, ни средства измерения.

В табл. 31-2 перечислены несколько возможных целей совершенствования процессов по работе с требованиями. Для краткости мы опустили суффикс вида «на X <количество> за Y <период времени>», который должен присутствовать в каждой цели. Для каждой цели в таблице предложены возможные индикаторы, которые позволят определить, оправдали ли себя вносимые изменения. Большинство измерений ПО являются запаздывающими индикаторами. Нужно время, чтобы новые подходы продемонстрировали устойчивое улучшение, поэтому новым процессам надо дать время, чтобы они укрепились и стали приносить пользу.

Табл. 31-2. Возможные ключевые показатели производительности для определенных целей улучшения процессов

Цель улучшения	Возможные индикаторы
Сократить объем переделок из-за ошибок в требованиях	Часы работы на всех стадиях жизненного цикла, относимые к ошибочным, нечетким, ненужным или отсутствующим требованиям.
	Процентная доля требований, в которых обнаружены ошибки после определения базовой версии
Снизить негативное влияние изменений в требованиях	Число новых требований, которые были представлены после определения базовой версии, но могли быть известными на момент ее определения.
	Процентная доля требований, которые изменились после определения базовой версии.
	Число часов в выпуске или итерации, потраченные на изменение результатов разработки из-за изменений в требованиях.
	Распределение запросов на изменение по источникам

Табл. 31-2. (*окончание*)

Цель улучшения	Возможные индикаторы
Сократить время, необходимое для выяснения требований в процессе разработки	Число проблем и вопросов, поднятых после определения базовой версии.
	Среднее время, необходимое для разрешения каждого вопроса или проблемы
Улучшить точность оценки общих усилий на разработку требований	Оценка и фактическое количество рабочих часов, потраченных на разработку требований в каждом выпуске и в целом в проекте
Сокращение числа реализованных, но ненужных функций	Процентная доля одобренных, но удаленных до реализации функций.
	Процентная доля функций, одобренных, но удаленных перед предоставлением клиенту выпуска или итерации

Если вы не уверены, какие индикаторы выбрать, воспользуйтесь приемом анализа, который называется «цель-вопрос-метрика» (goal-question-metric, GQM) (Basili и Rombach, 1988; Wiegers, 2007). GQM — это метод анализа в обратном направлении с целью определить, какие метрики будут полезными. Прежде всего сформулируйте цели улучшения. Для каждой цели сформулируйте вопросы, на которые нужно ответить, чтобы можно было судить о продвижении команды к цели. Наконец, определите метрики, которые дадут ответ на каждый вопрос. Эти метрики или их сочетание будут служить ключевыми показателями производительности.

Если вы выбрали реалистичные KPI для своих целей, но не видите признаков прогресса по истечении разумного времени, нужно провести расследование:

- Правильно ли были проанализированы проблемы и выявлены их первопричины?
- Выбрали ли вы действия по улучшению, непосредственно направленные на эти первопричины?
- Был ли реалистичным план реализации этих действий по улучшению? Был ли план реализован, как планировалось?
- Изменилось ли что-то со времени исходного анализа, что должно было заставить переориентировать действия команды по улучшению?
- Действительно ли члены команды приняли новые приемы работы и прошли период обучения, чтобы начать активно применять их на практике?
- Были ли поставлены реалистичные цели, которые команда была в состоянии достичь?

На пути к улучшению процессов работы с требованиями много ловушек — постарайтесь, чтобы проект по улучшению не застрял в одной из них.

Дорожная карта совершенствования работы с требованиями

Бессистемные подходы к совершенствованию процессов редко приводят к устойчивому успеху. Вместо того чтобы просто бросаться в бой, разработайте дорожную карту реализации улучшенных приемов работы с требованиями в вашей организации. Если вы уже пробовали один из методов оценки процессов работы с требованиями, описанных в этой главе, у вас уже есть некоторые идеи о приемах или образцах документов, наиболее полезных для вашей команды. Дорожная карта совершенствования процессов устанавливает такую последовательность действий по улучшению, которая дает наибольшую пользу при наименьших затратах.

Поскольку все ситуации разные, я не могу предложить вам один вариант дорожной карты на все случаи жизни. Стереотипные подходы к совершенствованию процессов не заменяют старательных размышлений и здравого смысла. На рис. 31-9 изображена дорожная карта совершенствования работы с требованиями одной организации. Желаемые бизнес-цели показаны в прямоугольниках в правой стороне рисунка, а основные действия по совершенствованию — в остальных прямоугольниках. Кружками отмечены промежуточные вехи (цели) вдоль путей достижения бизнес-целей (M1 обозначает веху за номером 1.) Реализуйте каждый набор действий по совершенствованию слева направо. Создав дорожную карту, назначьте ответственного за каждую веху, который впоследствии составит план действий по достижению этой цели. Затем превратите эти планы в действия!

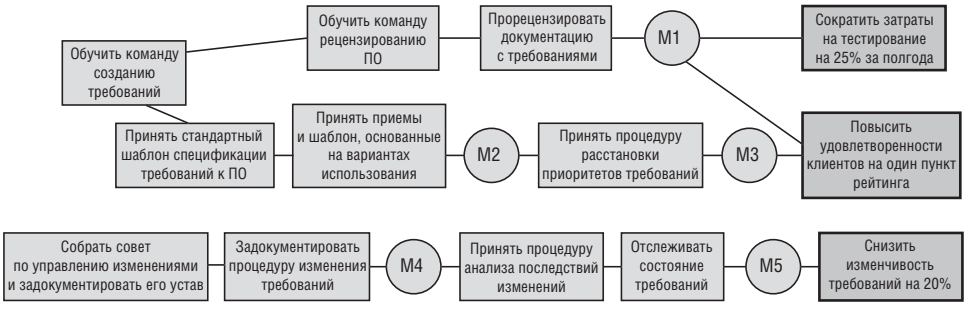


Рис. 31-9. Пример дорожной карты совершенствования процессов работы с требованиями

Что дальше?

- Выполните задание в Приложении А. Определите три улучшенных возможности для работы с требованиями, исходя из недостатков ваших текущих методов.
- Определите, какие из документов для процесса разработки требований, перечисленных на рис. 31-8, сейчас не используются в вашей организации, но были бы полезными.

- На основе двух предыдущих заданий разработайте дорожную карту совершенствования процессов работы с требованиями по образцу рис. 31-9. Убедите нескольких сотрудников вашей организации стать ответственными за каждую веху (цель). Попросите их на основе шаблона на рис. 31-6 написать план действий для достижения каждой вехи. Контролируйте процесс по мере реализации действий, указанных в планах.
- Выберите из этой книги один новый способ создания требований, о котором вы знаете недостаточно, и начните применять его со следующей недели — в буквальном смысле! Выберите два или три дополнительных способа и начните использовать их через месяц. Другие оставьте на потом, чтобы изучить и применять их через пять-шесть месяцев. Определите, когда вы хотели бы применять каждый из новых способов, преимущества, которые, как вы считаете, он даст вам, а также какая помощь или дополнительная информация может потребоваться. Подумайте, чье сотрудничество понадобится вам, чтобы начать использовать новые способы. Выявите препятствия, которые могут помешать применять конкретный способ, и подумайте, кто в силах помочь вам преодолеть их.

Глава 32

Требования к ПО и управление рисками

Дэйв, менеджер проекта Chemical Tracking System в компании Contoso Pharmaceuticals, встретился с ведущим программистом проекта Хелен и ведущим тестировщиком Рамешем. Все выразили радость по поводу новой совместной работы, но вспомнили и о проблемах, с которыми столкнулись в прошлом проекте под названием Pharm Simulator.

«Помните, как мы узнали, что пользователи терпеть не могут интерфейс Pharm Simulator только после бета-тестирования? — спросила Хелен. — У нас ушло четыре недели на то, чтобы переделать и заново протестировать его. Не хотела бы я еще раз пережить то ужасное время».

«Это было невесело,— согласился Дэйв. — Также раздражает, что пользователи, с которыми мы беседовали, клялись, что им нужно много функций, но до сих пор с ними никто не работает. На программирование моделирования взаимодействия препаратов потребовалось в три раза больше времени, чем мы ожидали, и в конце концов мы все равно исключили эту функцию. Сколько времени потрачено впустую!»

У Рамеша родилось предложение. «Может быть, нам нужно составить список проблем, возникших при работе над Pharm Simulator, чтобы попытаться избежать их при разработке Chemical Tracking System. Я читал статью по управлению рисками при создании ПО, где говорилось, что мы должны выявлять риски с самого начала и выяснять, как не дать им нанести вред проекту».

«Не уверен, что стоит, — возразил Дэйв. — Скорее всего, такие же проблемы не возникнут снова. Если мы выпишем все проблемы, возможные при разработке Chemical Tracking System, это будет выглядеть, как будто я не знаю, как вести проект. Я не хочу, чтобы в этом проекте кто-то думал о плохом. Мы должны планировать в расчете на успех!»

Как показывают последние высказывания Дэйва, программные инженеры и менеджеры проектов — вечные оптимисты. Мы часто ожидаем, что наш следующий проект пройдет гладко, несмотря на большое количество проблем в прошлых проектах. Реальность такова, что десятки возможных ловушек могут замедлить или пустить под откос проект по разработке ПО. Вопреки убеждениям Дэйва, в команде по разработке ПО *должны* выявлять и управлять рисками, начиная с тех, что связаны с требованиями.

Риск — это условие, которое может повлечь какие-либо потери или другим способом поставить под угрозу успех проекта. Это условие еще не породило проблему, и нужно, чтобы так оно и оставалось. Эти потенциальные проблемы могут оказать неблагоприятное воздействие на стоимость, сроки, технический успех, качество продукта или эффективность работы команды. *Управление рисками* — это процесс выявления, оценки и управления рисками до того, как они нанесут ущерб вашему проекту. Если что-либо нехорошее уже произошло с вашим проектом, то это — проблема, а не риск. Решайте текущие проблемы и вопросы, постоянно контролируя состояние проекта и процессы введения поправок.

Так как никто не может уверенно предсказывать будущее, управление риском — это способ минимизации вероятности проблем или ущерба от них. Управление рисками означает работу над опасностью до того, как она реализуется. Это повышает вероятность успеха проекта и уменьшает финансовые или другие последствия риска, которого не удастся избежать. Риск, лежащий вне сферы компетенции команды, следует передать для принятия мер руководству соответствующего уровня.

Поскольку требования так важны в проектах по созданию ПО, предусмотрительный менеджер уже на ранних стадиях проекта выявит риски, связанные с требованиями, и будет достаточно агрессивно контролировать их. Распространенные риски, связанные с требованиями, включают неверное понимание требований, недостаточное вовлечение пользователей, неточности или изменения в масштабах и целях проекта, и постоянно изменяющиеся требования. Менеджеры проектов могут управлять риском, связанным с требованиями, только в сотрудничестве с клиентами или их представителями. Совместное документирование рисков, связанных с требованиями и планирование действий по предотвращению их последствий укрепляет партнерство между клиентом и разработчиком, о котором говорилось в главе 2.

Просто знать о рисках недостаточно, чтобы они исчезли, поэтому в этой главе приводится краткий курс обучения управлению рисками при создании ПО (Wiegers, 2007). Позже в этой главе я также перечислю несколько рисков, которые могут поднять свои уродливые головы во время создания требований. Используйте эту информацию, чтобы разобраться с рисками, связанными с требованиями, до того, как они атакуют ваш проект.

Основы управления рисками при создании ПО

Проекты подстерегает масса рисков, помимо тех, что связаны с границами проекта и требованиями. Один из них — зависимость от внешних сущностей, таких как субподрядчик или другой проект, предоставляющий компоненты для повторного использования. При управлении проектом на первое место выходят такие риски, как неточные оценки, неприятие менеджерами точных оценок, недостаточная прозрачность состояния проекта и текучка кадров. Технологический риск представляет угрозу для очень сложных или исполь-

зующих передовые разработки проектов. Недостаток знаний — еще один источник таких рисков, как и исполнители, не обладающие достаточным опытом работы с используемыми технологиями или областью применения приложения. Переход на новые методы разработки создает целый букет новых рисков. А постоянно изменяющиеся нормы законодательства могут разрушить и самые лучшие планы проектов.

Именно поэтому во всех проектах нужно серьезно воспринимать управление рисками. Это значит, что необходимо исследовать горизонт в поисках айсбергов, а не нестись вперед на всех парах, веруя в непотопляемость своего корабля. Как и в случае с другими процессами, соотносите действия по управлению риском с масштабами своего проекта. Для небольших проектов достаточно простого списка рисков, а ключевым элементом успешного крупномасштабного проекта считается структурированное планирование управления рисками.

Составляющие управления рисками

Управление рисками (*risk management)* — это применение инструментальных средств и процедур для ограничения рисков в проекте приемлемыми рамками. Управление рисками предоставляет стандартный подход к выявлению и документированию рисков, оценке их возможного ущерба и предлагает стратегии их смягчения (Williams, Walker, и Dorofee, 1997). Управление рисками включает в себя действия, показанные на рис. 32-1 (адаптированная версия из источника [McConnell, 1996]).

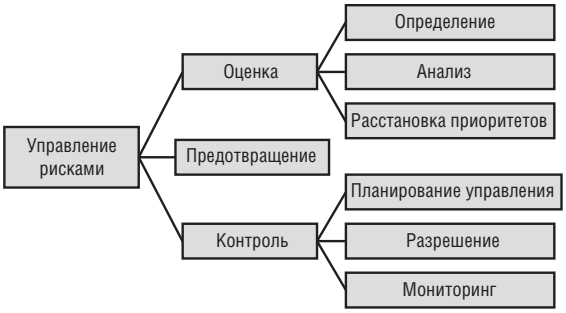


Рис. 32-1. Составляющие управления рисками

Оценка риска — это процесс исследования проекта для выявления возможных опасностей. Облегчите задачу *выявления рисков*, составляя списки рисков, типичных для разработки ПО, в том числе связанных с требованиями, описанными в разделе «Риск, связанный с требованиями» этой главы (Carr и др., 1993; McConnell, 1996). При *анализе риска* вы исследуете возможные последствия конкретных рисков для вашего проекта. Определение приоритетов рисков поможет сконцентрироваться на наиболее опасных, оценивая *подверженность* каждому из них. *Подверженность риску* — это функция, включающая как вероятность потерпеть урон из-за риска, так и масштабы этого урона.

Предотвращение риска — один из способов работы с риском: не делайте того, что рискованно. Вы можете избежать некоторых рисков, не начиная определенные проекты, полагаясь на испытанные, а не самые передовые технологии или исключая функции, которые будет особенно трудно воплотить верно. Однако разработка ПО рисковое предприятие, так что избавление от риска означает потерю возможности.

Большую часть времени вам придется управлять рисками, чтобы контролировать высокоприоритетные риски. Планирование управления рисками подразумевает создание плана действий для каждого значительного риска, включая методы смягчения, планы обеспечения непрерывности работы, ответственных лиц и сроки исполнения. Цель действий по смягчению воздействия рисков — либо не позволить риску стать проблемой, либо уменьшить его вредное воздействие. Риски не будут контролировать себя сами, поэтому *разрешение рисков* подразумевает реализацию планов по сдерживанию каждого риска. И, наконец, отслеживайте свое продвижение к разрешению каждого риска посредством *мониторинга риска*, который должен стать частью стандартной процедуры отслеживания состояния проекта. Отслеживайте, насколько хорошо работают ваши действия по смягчению риска, обнаруживайте новые риски, убирайте риски, угроза которых миновала, и периодически обновляйте приоритеты в своем списке рисков.

Документирование рисков проекта

Недостаточно просто знать риски, угрожающие проекту — нужно управлять ими так, чтобы иметь возможность сообщать о проблемах и состоянии рисков заинтересованным лицам на протяжении проекта. На рис. 32-2 показан шаблон документирования одного риска. Иногда эту информацию удобно хранить в электронной таблице или базе данных, что позволяет легко сортировать риски. Ведите список рисков в документе, отдельном от планов проекта, — так их будет легко обновлять на протяжении проекта.

Используйте формат «причина–следствие», документируя риски. То есть сначала формулируйте причину возникновения риска, вызывающую озабоченность, а затем ее возможный неблагоприятный результат — следствие. Часто люди, говорящие о риске, приводят только условие («клиенты не придут к единому мнению относительно требований к продукту») или только следствие («мы сможем удовлетворить лишь одного из наших основных клиентов»). Одна причина может повлечь несколько следствий, и несколько причин могут привести к одному и тому же следствию.

В шаблоне предусмотрены поля для записи о вероятности материализации риска в виде проблемы, о негативном влиянии на проект этой проблемы и об общей подверженности проекта риску. Я оцениваю вероятность по шкале от 0,1 (практически невозможно) до 1,0 (обязательно произойдет), а ущерб — по относительной шкале от 1 (нет проблем) до 10 (полный кошмар). Еще лучше попытаться оценить потенциальное влияние в единицах потерян-

ного времени или денег. Умножьте вероятность на влияние, чтобы оценить уязвимость для каждого риска.

Идентификатор:
<номер по порядку>

Автор:
<лицо, привлекшее внимание команды к риску>

Дата открытия:
<дата обнаружения риска>

Дата закрытия:
<дата закрытия риска>

Описание:
<описание риска в форме «причина–следствие»>

Масштаб влияния:
<проектные команды, бизнес- и функциональные области, на которые может оказать влияние риск>

Вероятность:
<вероятность того, что риск станет проблемой>

Влияние:
<оценка в баллах урона, который может быть нанесен, если риск станет проблемой>

Подверженность:
<вероятность, умноженная на ущерб>

План управления риском:
<один или больше методов управления, избегания, уменьшения последствий или других способов минимизации риска>

План обеспечения непрерывности работы:
<порядок действий на случай неэффективности плана управления риском>

Ответственное лицо:
<человек, отвечающий за разрешение риска>

Срок выполнения:
<дата, к которой надо выполнить действия по смягчению риска>

Рис. 32-2. Шаблон отслеживания риска

Не надо пытаться оценить риски слишком точно. Ваша цель — отделить наиболее опасные риски от тех, за которые не нужно хвататься немедленно. Проще оценивать и вероятность, и ущерб по шкале «высокий-средний-низкий». Риски, имеющие как минимум одну оценку *высокого* уровня, требуют внимания на ранних стадиях.

В поле «План управления риском» записывайте действия, которые планируется предпринять для контроля риска. Некоторые стратегии смягчения предусматривают снижение вероятность риска, другие — уменьшение его влияния. Учитывайте стоимость этих действий при планировании. Неразумно тратить 20 тыс. долларов на управление риском, реализация кото-

рого обошлась бы лишь в 10 тыс. Можно также составить планы на непредвиденные обстоятельства для самых угрожающих рисков, заранее решив, какие действия предпринять, если, несмотря на ваши усилия, риск в проекте реализуется. Назначьте для каждого риска, которым намереваетесь управлять, ответственное лицо и определите сроки исполнения действий по смягчению последствий. Долгосрочные или сложные элементы риска могут потребовать ступенчатой стратегии минимизации риска с массой промежуточных целей.

Идентификатор: 1 **Автор:** Юхонг Ли

Дата открытия: 22.08.13 **Дата закрытия:** (открыто)

Описание:
Недостаточное вовлечение пользователей в составление требований может вызывать необходимость объемной переработки пользовательского интерфейса после бета-тестирования.

Масштаб влияния:
Может оказывать влияние на всю систему, включая все пользовательские дополнения, разработанные для интегрированных компонентов серийной системы.

Вероятность: 0,6 **Влияние:** 7 **Подверженность:** 4.2

План управления риском:
1. Собрать требования по удобству использования на ранних стадиях выявления требований.
2. Провести встречи со сторонниками продукта для разработки требований.
3. Разработать одноразовый прототип базовой функциональности на основе информации от сторонников продукта.
4. Обеспечить оценку прототипа пользователями нескольких типов.

План обеспечения непрерывности работы:
Привлечь специалиста по удобству использования, чтобы проверить соответствие пользовательского интерфейса рекомендациям по дизайну интерфейсов человека и компьютера.

Ответственное лицо: Рейна Кабатана **Срок выполнения:** 13.02.14

Рис. 32-3. Пример отслеживания риска для Chemical Tracking System

Рис. 32-3 иллюстрирует риск, который лидеры проекта Chemical Tracking System обсуждали в начале этой главы. Команда оценила вероятность и ущерб на основании своего предыдущего опыта. Но, не оценив другие риски, она не узнает, насколько серьезна подверженность риску с коэффициентом 4,2. Первые два метода минимизации уменьшают вероятность того, что этот риск перейдет в проблему, за счет более активного привлечения пользователей к процессу составления требований. Создание прототипов снижает вероятность возможного ущерба, за счет получения отзывов пользователей на пользовательский интерфейс на ранних стадиях проекта.

Планирование управления рисками

Список рисков — не то же самое, что план управления рисками. В маленьком проекте можно включить планы управления рисками в план управления проектом разработки ПО. В большом проекте необходимо написать отдельный план управления рисками, формулирующий предполагаемые подходы для выявления, оценки, документирования и отслеживания рисков. Этот

план должен включать распределение ролей и обязанностей в действиях по управлению риском. Шаблон плана по управлению риском доступен для загрузки на веб-сайте книги. Во многих проектах назначается менеджер по управлению рисками, который отвечает за все, что может пойти не так. В одной компании такого сотрудника называли «Иа-Иа», в честь печального персонажа из «Винни-Пуха», который постоянно горевал о том, как все может быть плохо.

> **Внимание!** Не надо рассчитывать, что риск под контролем просто потому, что вы его обнаружили и выбрали действия по его смягчению. Эти действия нужно еще выполнить. Выделите достаточное время на управление рисками в расписании проекта, чтобы не потратить впустую усилия, затраченные на планирование управления рисками. Предусмотрите действия для смягчения риска, получение отчетов о состоянии риска и обновление списка рисков в списке задач по проекту.

Установите периодичность мониторинга рисков. Не упускайте из вида примерно десяток рисков с наибольшим коэффициентом подверженности и регулярно оценивайте эффективность методов по смягчению. Когда действие по минимизации завершено, заново оценивайте вероятность и влияние соответствующего риска, а затем в соответствии с результатами обновляйте список рисков и остальные текущие планы по смягчению. Риск не обязательно взят под контроль просто потому, что выполнены действия по его минимизации. Вам нужно понять, снизили ли эти методы подверженность до приемлемого уровня и не осталась ли возможность того, что один из рисков перерастет в проблему.

Без контроля

Менеджер проекта однажды спросил меня, что делать, если одни и те же пункты остаются в его еженедельном списке главных пяти рисков из недели в неделю. Это предполагает, что действия по минимизации воздействия этих элементов либо не исполняются, либо не эффективны. Если это так, то подверженность рискам, которые вы активно стараетесь взять под контроль, будет уменьшаться. В этом случае более значимыми и требующими вашего внимания становятся риски, представлявшие меньшую угрозу, чем пять главных. Периодически проводите переоценку вероятности материализации угрозы каждого элемента риска и потенциального ущерба от этого, чтобы понять, насколько результативны ваши действия по смягчению риска.

Риск, связанный с требованиями

Риски, описанные на следующих страницах, группируются по принадлежности к основным этапам создания требований: выявлению, анализу, спецификации, утверждению и управлению. В следующих разделах предлагаются методы, уменьшающие вероятность или влияние риска. Это только начало;

составьте свой собственный список рисков и стратегий по смягчению, на основе опыта, наработанного в каждом проекте. Терон Лейшман и Девид Кук (Leishman и Cook, 2002) описывают дополнительные риски, связанные с требованиями к ПО. Не забудьте записывать риски в формате «причина–следствие».

Выявление требований

Есть много факторов, препятствующих вашим усилиям по выявлению требований. Далее перечислены несколько областей возможных рисков при выявлении требований и представлены рекомендации по тому, как их избежать.

Концепция и границы проекта Расползание границ становится более вероятным, если у заинтересованных лиц проекта нет единого мнения о том, что должен (и не должен) представлять собой продукт. Начиная проект, составьте документ о концепции и границах, который содержит ваши бизнес-требования, и используйте его при принятии решений об утверждении или изменении требований.

Время, затраченное на разработку требований Жесткие сроки проекта часто заставляют менеджеров и клиентов пренебрегать составлением требований, потому что кажется, что если программисты не начнут работать над кодом немедленно, то не закончат вовремя. Записывайте, сколько усилий понадобилось в реальности на разработку требований к каждому проекту, чтобы иметь возможность оценить, достаточно ли этого и улучшить планирование следующих проектов. При использовании методов гибкой разработки (agile) программирование начинается раньше, чем в проектах с водопадным жизненным циклом.

Вовлечение клиента Недостаточное привлечение клиента к работе над проектом повышает шансы возникновения расхождений в ожиданиях. Определяйте заинтересованных лиц, клиентов и классы пользователей на ранних стадиях проекта. Определите, кто будет озвучивать пользователя в каждом из классов пользователей. Привлеките ключевых заинтересованных лиц в качестве сторонников продукта. Обеспечьте, чтобы сторонники продукта выполняли свои обязательства, а вы могли правильно выявить потребности.

Полнота и корректность спецификации требований Выявляйте пользовательские требования, которые соответствуют бизнес-потребностям, чтобы создать решение, которое даст клиентам ровно то, что им нужно. Составляйте конкретные сценарии использования, пишите тесты на основе требований и просите клиентов разрабатывать свои критерии приемки. Создавайте прототипы, чтобы требования были понятны пользователям и чтобы получать от них конкретные отзывы. Привлеките представителей клиентов к рецензированию спецификации требований и моделей анализа.

Требования к инновационным продуктам Легко ошибиться в оценке реакции рынка на продукты, первые в своем роде. Уделите большое внимание исследованию рынка, создавайте прототипы и используйте фокус-группы

пользователей, чтобы получать раннюю и частую обратную реакцию на ваш инновационный продукт.

Определение нефункциональных требований Поскольку основное внимание естественным образом обращено к функциональности продукта, легко упустить нефункциональные требования. Запрашивайте клиентов о качественных характеристиках, таких как производительность, легкость и простота использования, безопасность и надежность. Документируйте эти нефункциональные требования и критерии их принятия как можно точнее в спецификации требований к ПО.

Единство мнения клиентов относительно требований к продукту Если различные клиенты вашего продукта не достигли соглашения относительно того, что именно вы должны разрабатывать, кто-то их них останется недоволен результатом. Определите основных клиентов и используйте сторонников продукта для получения адекватного представительства и вовлечения клиентов. Удостоверьтесь, что вы полагаетесь на правильно выбранных людей, которые имеют полномочия для принятия решений. Привлеките соответствующих представителей заинтересованных лиц к рецензированию требований.

Несформулированные требования У клиентов часто есть скрытые ожидания, о которых они не сообщают, и поэтому те не документируются. Постарайтесь выявить любые допущения, которые может делать клиент. Используйте открытые вопросы, чтобы поощрить клиентов больше делиться своими мыслями, пожеланиями, идеями, информацией и сомнениями. Адресованные клиентам вопросы о том, что могло бы их заставить отвергнуть продукт, могут вскрыть темы, о которых вы еще не думали.

Использование существующего продукта в качестве основы требований Разработка требований считается не важной в проектах следующего поколения или при переделке предыдущих проектов. Разработчикам иногда рекомендуют использовать существующий продукт в качестве источника требований, со списком изменений и дополнений. Подробнее о некоторых способах обратного инжиниринга требований см. главу 21.

Решения, предлагаемые под видом потребностей Предлагаемые пользователями решения могут скрывать их действительные нужды, вести к автоматизации неэффективных бизнес-процессов и заставлять разработчиков принимать плохие архитектурные решения. Задача аналитика — докопаться до реальной потребности клиента, скрытой предлагаемым решением.

Отсутствие доверия между компанией и командой разработчиков Как вы уже видели в этой книге, для эффективной разработки требований требуется тесное сотрудничество многих заинтересованных лиц и особенно сообщества клиентов (бизнес-стороны ИТ-проекта) и разработчиков. Если эти команды не чувствуют, что их коллеги добросовестно трудятся для получения взаимовыгодного результата, возможны конфликты и выявление требований может оказаться под угрозой.

Анализ требований

Не очень осмотрительно записывать все, что скажет клиент, и сразу же погружаться в разработку. В анализе требований есть собственные зоны риска, которые описаны подробнее далее.

Определение приоритетов требований Удостоверьтесь, что каждое функциональное требование, функция или вариант использования получило приоритет и приписано к конкретной версии или итерации системы. Оцените приоритет каждого нового требования по отношению к объему оставшейся работы, чтобы иметь возможность принимать разумные решения о компромиссах и создавать планы итераций.

Технически сложные функции Оцените осуществимость каждого требования, чтобы определить те из них, воплощение которых может занять больше времени, чем ожидается. Отслеживайте состояние проекта, чтобы следить за требованиями, реализация которых отстает от графика. Старайтесь исправлять ситуацию как можно раньше. Для новых или рискованных требований создавайте прототипы, чтобы определить наиболее эффективные методики реализации.

Незнакомые технологии, методы, языки, инструменты или оборудование Не нужно недооценивать период обучения при работе с новыми методами, необходимыми для удовлетворения некоторых требований. Выявляйте эти рискованные требования в начале проекта и выделяйте достаточное время на фальстарты, обучение, и эксперименты.

Спецификация требований

В основе требований лежит взаимодействие. Тот факт, что требования изложены на бумаге или в письменном виде не означает, что они поняты.

Понимание требований Различия в интерпретации требований разработчиками и клиентами ведут к расхождениям в ожиданиях, когда произведенный продукт не способен удовлетворить нужды клиентов. Формальное рецензирование документации требований разработчиками, тестировщиками и клиентами уменьшают этот риск. Подготовленные, опытные бизнес-аналитики могут задать клиентам верные вопросы и составить высококачественные спецификации. Модели и прототипы, представляющие требования с разных точек зрения, также позволяют выявить неясные, неоднозначные требования.

Спешка, заставляющая действовать невзирая на незакрытые проблемы Полезно выделять области спецификации требований, которые требуют доработки специальными пометками, например «TBD», но рискованно начинать разработку, пока они не сняты. Записывайте имя человека, отвечающего за разрешение каждой неясности и планируемые сроки разрешения.

Неоднозначная терминология Создайте словарь для определения бизнес- или технических терминов, которые могут быть истолкованы разными читателями по-разному. Рецензирование спецификации требований поможет участникам выработать общее понимание ключевых терминов и понятий.

Включение дизайна в требования Описание дизайна в спецификации требований налагает ненужные ограничения на возможности, доступные разработчикам. А те сдерживают создание оптимального дизайна. Удостоверьтесь, что требования описывают, что необходимо сделать для решения бизнес-задачи, а не то, как это должно быть сделано.

Утверждение требований

Даже если вы хорошо потрудились над выявлением требований, важно подтвердить качество и верность решения, описанного в требованиях. С утверждением требований связаны следующие риски.

Неутвержденные требования Перспектива утверждения длинной спецификации требований пугает, как и мысль о написании тестов в самом начале процесса разработки. Тем не менее, если вы подтвердите корректность и качество требований до начала разработки, то сможете избежать значительной и дорогой переделки на более поздних стадиях проекта. Выделите время и ресурсы на эти действия в плане проекта. Добейтесь участия представителей клиентов в рецензировании требований. Также проводите пошаговые, неформальные рецензии, чтобы выявлять проблемы как можно раньше и дешевле.

Качество проверки Если проверяющие не знают, как проводить рецензирование документации требований, они могут упустить серьезные дефекты. Обучайте всех членов команды, которые будут участвовать в экспертизе. Пригласите человека с опытом проверки из вашей или сторонней организации для наблюдения и, возможно, координации ваших первых опытов, чтобы научить участников.

Управление требованиями

Большая часть рисков, связанных с требованиями, определяется порядком работы с изменениями. Эти и другие риски управления требованиями описаны ниже.

Изменение требований Вы можете уменьшить расползание границ проекта, используя документ о бизнес-требованиях и границах в качестве точек отсчета для утверждения изменений. Объединение усилий по выявлению требований с существенным участием пользователей может сократить расползание требований практически наполовину (Jones, 1996). Выявление ошибок в требованиях на ранних стадиях, сокращает количество последующих модификаций. Дизайн системы должен предусматривать простоту модификации, особенно при использовании итеративного цикла жизни.

Процесс изменения требований Риск, связанный с тем, как происходит изменение требований, зависит и от отсутствия определенного процесса изменений, использования неэффективных механизмов изменений и внесения коррективов без учета этого процесса. Очень важно, чтобы при изменении требований применялся анализ воздействия предлагаемых изменений, решения принимал совет по управлению изменениями и использовалось инстру-

ментальное средство для поддержки определенной процедуры. Жизненно важно ясное и четкое доведение изменений до соответствующих заинтересованных лиц.

Нереализованные требования Отслеживание требований помогает избежать пропуска каких-либо требований во время проектирования, разработки или тестирования.

Расширение границ проекта Если требования изначально плохо определены, дальнейшее их уточнение может увеличить объем проекта. Реализация нечетко определенных областей продукта потребует больше усилий, чем ожидалось. Ресурсы проекта, выделенные в соответствии с начальными неполными требованиями, могут оказаться недостаточными для удовлетворения полного спектра нужд пользователя. Для смягчения этого риска планируйте жизненный цикл, состоящий из поэтапных или разработанных средствами инкрементального моделирования версий. Реализуйте самую приоритетную функциональность в первом выпуске и наращивайте возможности системы в дальнейшем.

Управление рисками — ваш друг

Менеджер проекта может использовать управление рисками, чтобы выяснить условия, которые могут повлечь неблагоприятные последствия для проекта. Представьте себе менеджера нового проекта, озабоченного привлечением нужных пользователей в выявление требований. Если он умен, то сообразит, что это условие представляет собой риск, и внесет его в список рисков, оценив его вероятность и влияние, основываясь на предыдущем опыте. Если по прошествии некоторого времени пользователи все еще не вовлечены в работу над проектом, риск возрастет и, может быть, станет даже уже угрожать успеху проекта. Мне удавалось убеждать менеджеров отложить проект, когда не удавалось привлечь достаточно представителей пользователей, приводя такой аргумент: мы не должны впустую тратить деньги компании на заранее обреченный проект.

Периодическое отслеживание рисков позволяет менеджеру проекта знать масштабы угрозы от выявленных рисков. Сообщайте о недостаточно контролируемом риске вышестоящему руководству, которое может принять осознанное бизнес-решение исправить его либо продолжать работу, несмотря на риск. Управление рисками помогает вам держать глаза открытыми и принимать обоснованные решения, даже если вам не удается контролировать каждую неприятность, с которой может столкнуться ваш проект.

Что дальше?

- Определите несколько угрожающих вашему текущему проекту рисков, связанных с требованиями. Не обозначайте текущие проблемы как риски, отмечайте только то, что еще не случилось. Документируй-

те риски, используя шаблон на рис. 32-2. Предложите по крайней мере один возможный метод смягчения для каждого риска. Есть ли риски, с которыми вы готовы просто смириться в надежде, что они вас не заденут.

- Проведите мозговой штурм рисков с ключевыми заинтересованными лицами. Выявите как можно больше рисков, связанных с требованиями. Оцените каждый риск по вероятности его реализации и относительному влиянию и перемножьте эти величины для вычисления подверженности этому риску. Отсортируйте список по уменьшению подверженности, чтобы определить пять самых серьезных рисков, связанных с требованиями. Назначьте для каждого из них ответственного за исполнение действий по смягчению последствий реализации рисков.
- Постройте собственный список возможных рисков для требований в вашей организации. Начните с тех, что указаны в этой главе, и дополните список рисками, относящимися в вашему проекту. Этот расширенный список рисков позволит в будущем менеджеру проекта с самого начала иметь представление о возможных рисках.

Приложение А

Самостоятельная оценка применяемых приемов работы с требованиями

Это приложение содержит 20 вопросов, которые можно использовать для оценки своих текущих приемов работы с требованиями и определения областей, которые стоит усилить. Копия этого теста и электронная таблица для анализа результатов доступны для загрузки на веб-сайте книги. Есть более обстоятельные версии тестов, если нужно получить более точное понимание того, какие аспекты текущих приемов и документов больше всего выиграют от улучшений. Зайлефел (Seilevel, 2012) предлагает обширный тест проекта, который можно адаптировать для оценки приемов и результатов разработки требований в вашей организации.

Чтобы выполнить быструю оценку, выберите из четырех возможных вариантов ответа на каждый вопрос тот, который наиболее близко описывает то, как вы в настоящее время решаете конкретную проблему, связанную с требованиями. Если вы хотите оценить результаты количественно, присваивайте 0 баллов за каждый ответ «а», 1 балл за каждый ответ «б», 3 балла за каждый ответ «в» и 5 баллов за каждый ответ «г» (за исключением вопроса 16, в котором ответы «б», «в» и «г» оцениваются в 5 баллов каждый). Максимально возможное количество баллов — 100. В каждом вопросе указана ссылка на главу, где объясняется тема вопроса. Вообще говоря, чем выше оценка, тем более зрелые — и скорее всего более эффективные — ваши приемы работы с требованиями. В каждом вопросе указана глава или главы, в которых подробно описывается затронутая тема.

Не стремитесь получить как можно больше баллов — используйте этот тест, чтобы определить новые приемы, которые могут быть полезными в вашей организации. Не все вопросы подходят к типу разрабатываемого вами ПО. Кроме того, ситуации бывают разными; не для каждого проекта требуются скрупулезные подходы. Однако учитывайте, что неформальные подходы к работе с требованиями увеличивают вероятность значительных переделок. Большинство организаций получит выгоду от применения методов, представленных ответами «в» и «г».

На результаты может повлиять ваш выбор людей, выполняющих тест. Избегайте респондентов, которые вместо того, чтобы описать реальную ситуацию в организации, учитывают внутрикорпоративную политику, выдают желаемое за действительное или стараются дать те ответы, которые им кажутся «правильными». Выполнение самооценки независимо несколькими людьми позволит устранить часть этой тенденциозности и обеспечит более реалистичную картину ваших текущих приемов работы. Множественные респонденты также могут обнаружить различное понимание того, как определенные процессы выполняются в настоящее время. Вы можете воспользоваться электронной таблицей на веб-сайте книги для сбора разных ответов и изучения их распределения.

1. Как определяются, доводятся до людей и используются бизнес-требования проекта? [Глава 5]
   а. Иногда мы сначала пишем высокоуровневое описание продукта, но в дальнейшем мы к нему не возвращаемся.
   б. Человек, который знает продукт, владеет бизнес-требованиями и обсуждает их устно с разработчиками.
   в. Мы фиксируем бизнес-требования в документе о концепции и границах проекта, уставе проекта или аналогичном документе в соответствии со стандартным шаблоном. У всех заинтересованных лиц проекта есть доступ к этому документу.
   г. Мы активно используем задокументированные бизнес-требования в проекте, оценивая предлагаемые функции продукта и изменения требований на предмет их соответствия задокументированным границам и корректируем границы по мере необходимости на основе бизнес-целей.
2. Как выявляются и характеризуются группы пользователей продукта? [Глава 6]
   а. Разработчики догадываются, кто пользователи продукта.
   б. Отдел маркетинга или куратор проекта полагают, что знают, кто пользователи продукта.
   в. Целевые группы пользователей и сегменты рынка определяют менеджеры на основе исследования рынка, нашей существующей базы пользователей и информации от других заинтересованных лиц.
   г. Заинтересованные в проекте лица определяют четкие классы пользователей, характеристики которых кратко записаны в документации требований к ПО.
3. Как вы получаете информацию от клиентов о требованиях? [Глава 7]
   а. Разработчики уверены что знают, что создавать.
   б. Проводится анкетирование или опрос фокус-групп пользователей.
   в. Мы встречаемся с людьми индивидуально или иногда в группах, и они рассказывают нам о своих желаниях.

г. Используются различные приемы выявления требований, включая интервью и семинары с представителями классов пользователей, анализ документов и системного интерфейса.

4. Насколько хорошо обучены и насколько опытны ваши бизнес-аналитики? [Глава 4]
   а. Они — разработчики или бывшие пользователи, имеющие мало опыта и не проходившие обучение в области разработки требований к ПО.
   б. Разработчики, опытные пользователи или менеджеры проектов, имеющие некоторый опыт разработки требований, исполняют роль аналитиков.
   в. Бизнес-аналитики прошли обучение в течение нескольких дней и имеют значительный опыт сотрудничества с пользователями.
   г. У нас есть профессиональные бизнес-аналитики или разработчики требований, имеющие подготовку и опыт в области приемов проведения собеседований, ведения собраний групп, написания технических текстов и моделирования. Они знают как прикладную область, так и процесс разработки ПО.
5. Как высокоуровневые системные требования распределяются по программным частям продукта? [Главы 19 и 26]
   а. Предполагается, что ПО будет работать, несмотря на любые недостатки оборудования.
   б. Разработчики ПО и оборудования обсуждают, какие системы должны выполнять конкретные функции.
   в. Конструктор или архитектор системы анализирует системные требования и решает, какие из них будут реализованы в каждой подсистеме.
   г. Опытные члены команды распределяют части системных требований по программным подсистемам и компонентам и отслеживают их связи с отдельными требованиями к ПО. Интерфейсы подсистем явно определяются и документируются.
6. Насколько часто в ваших проектах происходит повторное использование требований? [Глава 18]
   а. Мы не используем требования повторно.
   б. Бизнес-аналитик, знакомый с предыдущими проектами, иногда знает, какие требования можно повторно использовать в новом проекте, поэтому он копирует и вставляет эти требования в новую спецификацию.
   в. Бизнес-аналитик может поискать в предыдущих проектах, которые хранятся в нашем инструменте управления требованиями, те требования, которые могут пригодиться в новом проекте. Он может повторно использовать отдельные версии этих требований, применяя встроенные функции инструмента.
   г. Мы создали хранилище требований, которые подходят для повторного использования, адаптированы и усовершенствованы из предыдущих

проектов. Бизнес-аналитики используют стандартную процедуру проверки хранилища на предмет требований, которые могут повторно использовать в их текущих проектах. Мы используем связи для отслеживания дочерних и зависимых требований, элементов дизайна и тестов при повторном использовании требования.

7. Какие приемы работы с заинтересованными лицами используются для выявления конкретных требований к ПО? [Главы 7, 8, 12 и 13]
   а. Мы начинаем с общего понимания, пишем часть кода, а потом модифицируем его, пока все не получится.
   б. Руководство или отдел маркетинга задает концепцию продукта, а разработчики пишут требования. Заинтересованные лица клиента сообщают разработчикам, если они что-то упустили.
   в. Представители отдела маркетинга или клиентов говорят разработчикам, какие функции и возможности должен содержать продукт. Иногда маркетологи сообщают разработчикам об изменении направления продукта.
   г. Мы проводим организованные собеседования или семинары по выявлению требований с участием представителей различных классов пользователей продукта. Мы применяем варианты использования и пользовательские истории, чтобы понять цели пользователей, и мы создаем модели анализа, чтобы убедиться, что выявили все функциональные требования. Мы формулируем требования постадийно и итеративно, предоставляя клиентам массу возможностей скорректировать их.
8. Как документируются требования к ПО? [Главы 10, 11, 12 и 30]
   а. Наши требования к ПО складываются из устных рассказов, электронных и голосовых сообщений и заметок, сделанных во время собеседований и семинаров.
   б. Мы составляем описания в неструктурированной повествовательной текстовой форме, создаем простые списки требований или рисуем какие-то диаграммы.
   в. Мы записываем требования на структурированном естественном языке в соответствии со стандартным шаблоном. Иногда мы дополняем эти требования графическими моделями анализа с применением стандартной нотации.
   г. Мы создаем требования и графические модели анализа и храним их все в коммерческом инструментальном средстве управления требованиями. Вместе с каждым требованием хранятся несколько его атрибутов.
9. Как выявляются и документируются такие нефункциональные требования, как атрибуты качества ПО? [Глава 14]
   а. А что такое «атрибуты качества ПО»?
   б. Мы проводим бета-тестирование, чтобы получить отзывы пользователей о том, насколько им понравился продукт.

в. Мы документируем некоторые атрибуты, такие, как производительность, простота использования и требования к безопасности.

г. Мы работаем с клиентами над выявлением важных атрибутов качества каждого продукта, которые затем документируем методом, обеспечивающим точность и возможность проверки.

10. Как идентифицируются отдельные функциональные требования? [Глава 10]

а. Мы пишем абзацы пояснительного текста или короткие пользовательские истории; конкретные требования не идентифицируются по отдельности.

б. Мы используем маркированные или нумерованные списки.

в. Мы используем иерархическую схему нумерации, например «3.1.2.4».

г. Каждое конкретное требование имеет свой уникальный, содержательный идентификатор, не изменяющийся при добавлении, перемещении или удалении других требований.

11. Как определяются приоритеты требований? [Глава 16]

а. Все требования важны, поэтому не нужно назначать им приоритеты.

б. Клиенты говорят нам, какие требования для них более важные. Если клиента нам этого не говорят или не согласны, тогда решают разработчики.

в. Клиенты приходят к единому мнению о разделении всех требований на категории высокого, среднего и низкого приоритета.

г. Мы принимаем решения о приоритетах при помощи аналитического процесса, посредством которого определяем ценность для клиента, стоимость и технический риск каждого требования, или используем похожий прием структурированной приоритизации.

12. Какие методы используются для подготовки частичного решения и подтверждения единого понимания проблемы? [Глава 15]

а. Мы просто создаем систему, а затем устраняем недостатки, если это потребуется.

б. Мы создаем несколько простых прототипов и просим пользователей дать свои отзывы. Иногда нас заставляют выпускать прототип как окончательный продукт.

в. Мы создаем прототипы и для имитации пользовательского интерфейса, и для технической проверки концепции, когда это целесообразно.

г. Наши планы по проектам включают задачи создания электронных или бумажных одноразовых прототипов для уточнения требований. Иногда мы создаем эволюционные прототипы. Мы используем сценарии оценки для получения отзывов клиентов на наши прототипы.

13. Как утверждаются требования? [Глава 17]

а. Мы думаем, что неплохо пишем требования с первого раза.

б. Мы даем почитать документацию требований разным людям, чтобы узнать их мнения.

в. Бизнес-аналитик и некоторые заинтересованные лица проводят неформальное рецензирование.

г. Мы проверяем свою документацию и модели требований, привлекая к этому клиентов, разработчиков и тестировщиков. Мы составляем тесты требований и применяем их для проверки требований и моделей.

14. Как различаются различные версии документации требований? [Глава 27 и 30]

а. Автоматически генерируется дата распечатки документа.

б. Мы используем последовательную нумерацию, например 1.0, 1.1 и т. д., для каждой версии документа.

в. У нас есть ручная схема идентификации, отделяющая черновые версии от основных и крупные изменения от мелких.

г. Версии документации требований хранятся в системе управления документами с поддержкой версий, либо требования хранятся в инструментальном средстве управления требованиями, ведущем историю изменений каждого требования.

15. Как отслеживается связь требований с источником? [Глава 29]

а. Никак.

б. Мы знаем, откуда появилось большинство требований, но не документируем эту информацию.

в. У каждого требования есть установленный источник.

г. У нас установлено двусторонние связи между бизнес-требованиями, системными требованиями, пользовательскими требованиями, функциональным и нефункциональным требованиями.

16. Как требования используются для разработки планов проекта? [Глава 19]

а. Дата выпуска продукта устанавливается до того, как мы начинаем разработку требований. Мы не можем изменить ни график проекта, ни требования. Иногда мы выполняем быструю корректировку границ проекта, при которой удаляем функции непосредственно перед датой выпуска.

б. Первая итерация плана работы над проектом устанавливает сроки сбора требований. Оставшаяся часть плана проекта разрабатывается после того, как получено предварительное понимание требований. Но после этого мы практически не можем изменить план.

в. Мы начинаем с информации о требованиях, которой достаточно для определения их приоритетов, после чего оцениваем усилия, необходимые для реализации самых приоритетных требований. Мы разрабатываем требования и ПО инкрементально, планируя требования на каждую итерацию, ориентируясь на их приоритет и размер. Если нам нужно реализовать больше требований, чем позволяет наш план, мы добавляем дополнительные итерации.

г. Мы вырабатываем сроки и планы на основе предполагаемых затрат, необходимых для реализации требуемой функциональности, начиная с самых приоритетных требований. Эти планы обновляются по мере изменения требований. Если нам нужно отказаться от каких-либо функций или изменить распределение ресурсов для выполнения обязательств по срокам, это делается как можно раньше. Мы планируем предоставлять несколько выпусков, чтобы учесть изменения и разрастание требований. [Примечание: «в» и «г» одинаково хорошие ответы на этот вопрос.]

17. Как требования используются в качестве основы для дизайна? [Глава 19]
    а. Если у нас есть записанные требования, мы обращаемся к ним при разработке.
    б. В документации требований описаны решения, которые мы намереваемся реализовать.
    в. Каждое функциональное требование отслеживается к элементу дизайна.
    г. Дизайнеры проверяют, может ли спецификация требований служить основой для дизайна. Мы поддерживаем полную двустороннюю связь между отдельными функциональными требованиями и элементами дизайна.
18. Как требования используется в качестве основы для тестирования? [Глава 19]
    а. Тестировщики тестируют ПО, основываясь на своем понимании того, как оно должно работать.
    б. Тестировщики проверяют то, что, по словам разработчиков, реализовано.
    в. Мы составляем тесты системы на основе пользовательских и функциональных требований.
    г. Тестировщики проверяют, поддаются ли требования, документированные в спецификации, проверке, и создают планы тестирования. Мы отслеживаем связи системных тестов системы с конкретными функциональными требованиями. Один из параметров измерения прогресса тестирования системы — по охвату требований.
19. Как определяется базовая версия требований для каждого проекта и как ею управляют? [Главы 2 и 27]
    а. У нас нет базовой версии, потому что в нашем проекте используется гибкая разработка (agile).
    б. Клиенты и менеджеры объявляют требования законченными, но разработчикам все равно приходит много запросов на изменения и жалоб.
    в. Мы определяем в спецификации первоначальную базовую версию требований, но не всегда вовремя обновляем ее по мере внесения изменений.

г. При определении базовой версии требования хранятся в средстве управления требованиями. Хранилище требований обновляется каждый раз, когда утверждаются изменения в требованиях. Мы ведем учет истории изменений каждого требования, внесенного в базовую версию. В проектах гибкой разработки в команде согласовывают базовую версию требований для каждой итерации.

20. Как управляют изменениями в требованиях? [Глава 28]

а. Требования меняются всякий раз, как у кого-то появляется новая идея или когда кто-то понимает, что он что-то забыл.

б. Мы боремся с изменениями, замораживая требования после завершения фазы разработки требований, но все равно неформальные соглашения об изменениях имеют место.

в. Мы используем утвержденный формат и центральный пункт подачи запросов на изменения. Менеджер проекта решает, какие изменения утвердить.

г. Изменения вносятся согласно нашему документированному процессу управления изменениями. Мы используем инструментальное средство для сбора, хранения и передачи запросов на изменения. Последствия каждого изменения оцениваются прежде, чем совет по управлению изменениями принимает решение о его одобрении.

Приложение Б

Руководство по разрешению проблем с требованиями

Действуя настойчиво, при взаимодействии различных заинтересованных в проекте лиц, вы можете успешно реализовать усовершенствованные приемы разработки и управления в своей организации. Следует выбрать приемы, которые решат или предотвратят те или иные проблемы, связанные с требованиями, возникающими в ваших проектах. После определения наиболее срочных проблем важно выявить первопричины каждой такой проблемы. Эффективные решения обращены к первопричинам, а не только к видимым на поверхности симптомам.

В этом приложении перечислено множество сгруппированных по нескольким категориям симптомов проблем разработки требований, с которыми вы можете столкнуться. Симптомы дополнены описаниями соответствующих возможных первопричин и предложениями по решению каждой проблемы. Конечно, это не единственные проблемы, которые могут быть связаны с требованиями, так что дополняйте эту таблицу своими собственными примерами, которые будут появляться у вас по мере того, как вы будете обнаруживать симптомы, не указанные здесь, — и разрешать их. Иногда наблюдаемые симптомы сами являются первопричинами других проблем. Например, симптом «Бизнес-аналитики не знают, как качественно справиться со своей ролью» является первопричиной множественных наблюдаемых симптомов проблем с выявлением требований. Эти вещи связываются в цепочки, и здесь показаны не все возможные связи.

К сожалению, нет гарантии, что предложенные решения излечат ваши конкретные симптомы, особенно если вызывающие их проблемы связаны с корпоративной политикой или культурой или если первопричины лежат вне сферы действия команды разработчиков. И, как мы говорили ранее, ни одно из этих решений не поможет, если вы имеете дело с неразумными людьми.

Общие симптомы проблем, связанных с требованиями

Проблемы — это условия, которые ведут к негативным последствиям для проекта. Вот некоторые признаки, указывающие на возможность проблем с требованиями в проекте:

- продукт не удовлетворяет нужд пользователей или не соответствует их ожиданиям;
- продукт требует исправлений и «заплат» сразу после выпуска;
- выпущенное решение не помогает организации в достижении бизнес-целей;
- нарушение графика и перерасход бюджета;
- разочарование членов команды, деморализация, утрата мотивации и текучка кадров;
- масса переделок во время разработки решения;
- упущенная возможность на рынке или задержка с получением бизнес-преимущества;
- потеря доли на рынке или доходов;
- возврат продукта, неприятие продукта рынком или плохие отзывы о продукте.

Стандартные препятствия для реализации решений

Любая попытка изменить способы работы отдельных людей или организаций встречает сопротивление. Определив, как исправить первопричины ваших проблем с требованиями, подумайте также о препятствиях, которые могут затруднить реализацию этих действий, и о возможных путях их обхода. Вот некоторые типичные препятствия на пути к реализации изменений в приемах работы с требованиями:

- недостаточное признание или понимание проблем, вызываемых текущими приемами работы с требованиями;
- нехватка времени — все и так слишком заняты);
- давление рынка, диктующее быстрейший выпуск продукта;
- недостаточная заинтересованность руководства в затратах на разработку требований;
- скептицизм относительно ценности разработки требований;
- нежелание принимать новый или более упорядоченный процесс разработки требований или ПО;
- политика и устоявшаяся корпоративная культура;
- трения между заинтересованными лицами;
- недостаточная подготовка и обучение персонала;
- неясное распределение ролей и обязанностей в проекте;

- недостаточное число ответственных лиц и плохая отчетность за действия, связанные с требованиями.

Обратите внимание, что это вопросы, связанные с людьми или передачей информации, а не технические препятствия. Большинство препятствий такого рода не удается преодолеть легко и быстро, но первый шаг — признать их наличие.

Руководство по устранению проблем, связанных с требованиями

Чтобы использовать этот раздел, выявите симптомы, которые указывают, что действия, связанные с требованиями, идут в вашем проекте не так хорошо, как хотелось бы. Найдите в колонке «Симптомы» что-либо, похожее на обнаруженное вами. Также в колонке «Симптомы» можно поискать условия, применимые к вашему проекту или организации. Затем изучите колонку «Возможные первопричины», чтобы понять, какие факторы могут вызывать проблемы в вашей среде. Далее выберите приемы и подходы из колонки «Возможные решения», которые, по вашему мнению, эффективно подействуют на первопричины и, если все пойдет хорошо, позволят решить проблему.

Проблемы, связанные с процессами

Описанные в этом разделе симптомы говорят о необходимости корректировки процессов разработки и управления требованиями.

Симптомы	Возможные первопричины	Возможные решения
Процессы работы с требованиями и шаблоны документов в различных проектах несовместимы между собой Используемые процессы работы с требованиями не эффективны Шаблоны документов незрелые или используются неправильно	Отсутствие общего понимания процессов работы с требованиями Отсутствие механизма совместного использования шаблонов и документации процессов Отсутствие хороших примеров шаблонов и документации требований Не определены процессы работы с требованиями Бизнес-аналитики не понимают, как правильно использовать все разделы шаблонов	Используйте ретроспективу проектов, чтобы выяснить текущие проблемы и их последствия Задокументируйте текущий процесс работы с требованиями и создайте описание-предложение желаемого процесса Обучите всех членов команды созданию требований Примите один или несколько стандартных шаблонов документов требований. Обеспечьте инструкции по подгонке шаблонов к конкретным проектам. Соберите и обеспечьте доступ к хорошим примерам документации требований в общедоступном хранилище Выясните, не слишком ли сложны шаблоны для всех проектов, и по возможности упростите их

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Люди с ролью бизнес-аналитика не знают, как выполнять ее качественно	Недостаток знаний или опыта создания требований и выполнения роли бизнес-аналитика Руководство считает, что любой пользователь, разработчик или другой член команды автоматически может стать хорошим бизнес-аналитиком, поэтому люди назначаются на эту роль без обучения и руководства	Обучите будущих бизнес-аналитиков как созданию требований, так и соответствующим навыкам взаимодействия с людьми Напишите должностную инструкцию и список необходимых навыков для бизнес-аналитиков Организуйте программу обучения новых аналитиков Предоставьте руководству описания ролей бизнес-аналитика в разных организациях Выработайте перспективы карьеры в организации для бизнес-аналитика
Средства управления требованиями используются недостаточно	Недостаточное обучение использованию возможностей инструментального средства Процессы и культура работы не изменены для извлечения максимальной выгоды из инструментальных средств Нет ответственных за внедрение инструментального средства Недооценено время, необходимое на конфигурирование, изучение и развертывание средства	Отправьте нескольких бизнес-аналитиков на тренинг, организованный производителем инструментального средства. Определите сторонника инструментального средства для обеспечения его внедрения и помощи другим пользователям Выявите и измените особенности процессов и культуры, препятствующие использованию инструментального средства в полной мере

Проблемы, связанные с продуктом

Некоторые проблемы с продуктами указывают на необходимость совершенствования работы с требованиями.

Симптомы	Возможные первопричины	Возможные решения
Недовольные клиенты Клиенты отвергают продукт Плохие отзывы о продукте Низкие продажи, потеря доли рынка Получение огромного числа запросов на изменение	Недостаточное вовлечение пользователей в разработку требований Нереалистичные ожидания клиентов Несоответствие между пониманием конкретных требований клиентами и разработчиками Недостаточное исследование рынка Неудачная формулировка проблемы Необходимые изменения не внедряются во время разработки Разработчики создавали продукт на основе своих догадок, а не четко определенных требований	Определите классы пользователей Определите сторонников продукта Соберите фокус-группы Используйте методы совместной работы для выявления требований Создавайте прототипы и просите пользователей оценивать их Привлеките представителей клиентов к участию в рецензировании документации требований Применяйте поэтапные и итеративные методы разработки для адаптации к потребностям клиентов
Продукт не удовлетворяет бизнес-целей	Отсутствие ясных и точных бизнес-требований, в том числе бизнес-целей и метрик успеха	Разработка бизнес-требований с участием ключевых заинтересованных лиц Понимание, какие метрики успеха важны для заинтересованных бизнес-лиц проекта Доведение бизнес-целей до других заинтересованных лиц с целью достижения единства мнений

Проблемы, связанные с планированием

Перечисленные в этом разделе симптомы указывают на неоптимальность согласования планирования требований и проекта.

Симптомы	Возможные первопричины	Возможные решения
Требования не полны Требования недостаточно детализированы Разработка следующего выпуска начинается до того, как требования для него поняты в достаточной мере	Недостаточное вовлечение пользователей в разработку требований Разработке требований уделено недостаточно времени Дата выпуска версии определена до достижения полного понимания требований Ключевые маркетологи или заинтересованные лица бизнес-подразделений не вовлечены в процесс работы над требованиями Руководство или клиенты не понимают необходимости в требованиях У аналитиков и разработчиков нет единого мнения о содержании требований Отслеживание требований не используется для обнаружения пробелов Слишком много открытых проблем с требованиями	Не принимайте решений о сроках выпуска до того, как требования станут понятными в достаточной мере В проекте гибкой разработки будте готовы сокращать границы продукта или добавлять итерации при уточнении требований Привлекайте разработчиков На ранних стадиях, чтобы они понимали требования Тщательно определяйте бизнес-требования, особенно границы продукта Объясните заинтересованным лицам риски, связанные с поспешной разработкой Добивайтесь сотрудничества между бизнес-аналитиками, разработчиками и бизнес-партнерами, чтобы те ставили реалистичные цели Используйте приемы инкрементального моделирования, чтобы быстро начать выпускать продукт, приносящий пользу клиенту Разработчики должны рецензировать требования прежде, чем воплощать их Отслеживайте связи функциональных требований с пользовательскими и бизнес-требованиями для выявления пропущенных требований Управляйте и отслеживайте состояние проблем с требованиями

Симптомы	Возможные первопричины	Возможные решения
Сокращение сроков после начала проекта, но без сокращения объема проекта	Заинтересованные лица не понимают влияния сокращения сроков на реализацию проекта в полном объеме	Добивайтесь сотрудничества между бизнес-аналитиками, разработчиками и бизнес-партнерами над определением реалистичных целей Договаривайтесь о компромиссах при изменении ограничений проекта Используйте более качественные приемы оценки
Часть необходимой и запланированной работы над требованиями не выполняется Несколько человек совершают одни и те же операции с требованиями	Нечеткое определение ролей и обязанностей при создании требований Задачи по работе над требованиями не включены в планы проекта Нет ответственного за управление требованиями	Определите роли и обязанности при создании требований в каждом проекте Выделите необходимые ресурсы для эффективной разработки и управления требованиями Вносите действия, связанные с требованиями, в планы и графики работы по проектам
К реализации планируется больше требований, чем позволяют сроки и ресурсы	График устанавливается до того, как определены требования Готовность выполнить проект подтверждается до тщательной оценки его объема Неконтролируемое расширение границ проекта Не учитывается период обучения незнакомым технологиям или работе с новыми инструментальными средствами Для участия в проекте недостаточно персонала Заинтересованные лица опасаются, что у них будет только один выпуск, в котором нужно реализовать все необходимое	Прежде чем принимать обязательства по проекту, задокументируйте концепцию и границы продукта в соответствии с бизнес-целями График разработки составляйте на основе требований Планируйте несколько циклов выпуска с расчетом на позднюю реализацию низкоприоритетных требований Включите время на подготовку и период обучения в график работы Определите приоритеты требований на основе бизнес-целей Ставьте жесткие временные рамки для разработки или выпускайте функции продукта поэтапно Динамически корректируйте приоритеты проекта, насколько это диктует ситуация с проектом

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Недокументированный или плохо определенный объем проекта Плохое планирование выпусков или итераций	Неясные бизнес-цели Спешка с началом разработки Отсутствие понимания важности определения объема проекта Отсутствие единства мнений у заинтересованных лиц об объеме проекта Изменчивые условия рынка или быстро изменяющиеся бизнес-потребности	Не начинайте проект без четких бизнес-целей Задокументируйте концепцию и границы проекта и согласуйте их с ключевыми заинтересованными лицами Откладывайте или отменяйте проект, если отсутствует поддержка и не определен объем проекта Используйте более короткие итерации разработки, чтобы гибко подстраиваться к быстрым изменениям требований

Проблемы, связанные с обменом информацией

Многие проблемы, включая перечисленные в следующей таблице, возникают из-за неэффективного общения между заинтересованными лицами проекта.

Симптомы	Возможные первопричины	Возможные решения
Дублирование действий, когда несколько человек реализуют одни и те же требования	Обязанности по реализации требований неясны Недостаточная коммуникация между подгруппами, работающими над частями проекта	Четко определите роли и обязанности по реализации ПО Обеспечьте видимый контроль состояния каждого требования Введите более эффективные методы коммуникации и приемы работы среди членов команды
Пересмотр принятых ранее решений	Отсутствие ясного признания и наделения полномочиями людей, принимающих соответствующие решения Невозможность зафиксировать, как и почему принимаются решения	Определите ответственных за принятие решений по требованиям к данному проекту и процедуру принятия решений Определите и наделите полномочиями сторонников продукта Задокументируйте, почему в прошлом требования отвергались, откладывались или отменялись

Симптомы	Возможные первопричины	Возможные решения
Вопросы и проблемы с требованиями не разрешаются	Отсутствие порядка работы с вопросами и проблемами, возникающими с требованиями Обязанности по разрешению проблем неясны Никто не отвечает за отслеживание проблем и их состояние В команде не могут получить необходимую информацию от поставщика, клиента, подрядчика и другого заинтересованного лица	Назначайте каждую открытую проблему для разрешения конкретным людям Используйте средство управления дефектами для отслеживания проблем с требованиями до их закрытия Отслеживайте открытые проблемы в рамках общего мониторинга проекта С самого начала добейтесь согласия всех заинтересованных лиц на открытый и своевременный обмен информацией, а также на предоставление ответов и разрешение проблем
Участники проекта используют разную терминологию	Предположение, что все толкуют ключевые термины одинаково и верно	Определите значения терминов в словаре Определите термины, касающиеся структур данных, в словаре Организуйте обучение команды разработчиков особенностям соответствующей предметной области Обучите представителей пользователей созданию требований

Проблемы, связанные с выявлением требований

Многие симптомы указывают, что члены команды, задействованные в выявлении требований, работают не столь эффективно, как могли бы.

Симптомы	Возможные первопричины	Возможные решения
Команда не может добиться достаточного вовлечения представителей клиентов в выявление требований Разработчики делают большое количество допущений о реализуемых функциях Разработчикам приходится разрешать возникающие вопросы по требованиям	У представителей клиентов недостаточно времени для участия в разработке требований Клиенты не понимают необходимости своего участия Клиенты не знают, что бизнес-аналитикам от них нужно Клиенты не заинтересованы в проекте Клиенты думают, что разработчики должны и так знать, что нужно клиентам Бизнес-аналитики не знают свою клиентскую аудиторию У бизнес-аналитиков нет доступа к реальным клиентам Сопротивление соблюдению процесса разработки требований На проект не назначено ни одного бизнес-аналитика	Объясните клиентам и менеджерам особенности разработки требований и объясните необходимость их участия Опишите клиентам и менеджерам риски, возникающие из-за недостаточного вовлечения пользователей Создайте атмосферу сотрудничества между разработчиками и клиентами Определите классы пользователей или сегменты рынка Определите сторонников продукта в каждом классе пользователей Заинтересуйте руководителей разработчиков и клиентов в эффективности процесса разработки требований Четко определите роли и обязанности Регулярно проводите встречи с клиентами с конкретной повесткой дня
Вовлечены не те представители пользователей	Менеджеры, маркетологи, или другие лица пытаются говорить за конечных пользователей Менеджеры не обеспечили доступность пользователей бизнес-аналитикам	Определите классы пользователей Определите и наделите полномочиями соответствующих и эффективных сторонников продукта Разработайте архетипы пользователей, служащие заменой реальным пользователям Отклоняйте запросы на добавление требований из неуполномоченных или ненадлежащих источников

Симптомы	Возможные первопричины	Возможные решения
Пользователи не уверены в своих потребностях	Пользователи не понимают или не могут хорошо описать бизнес-процессы Система создается для поддержки нового, не полностью определенного бизнес-процесса Пользователи не заинтересованы в проекте, может быть, боятся его Бизнес-цели определены или доводятся недостаточно качественно	Проясните, какие результаты ожидают заинтересованных лиц, которых это затрагивает, в случае успеха проекта Выявите сторонников или владельцев продукта Смоделируйте пользовательские бизнес-процессы Разработайте план выявления требований, чтобы определить источники требований и выбрать соответствующие приемы выявления требований Сформулируйте список общих вопросов в качестве отправной точки для работы по выявлению требований Разработайте варианты использования или пользовательские истории Создавайте прототипы и просите пользователей оценивать их Используйте инкрементальный подход к разработке, чтобы прояснять требования понемногу за раз
Менеджер проекта или бизнес-аналитик не знают, кто их пользователи	Плохо определенная концепция продукта Плохо понимаемые потребности рынка	Задокументируйте концепцию проекта Проведите исследование рынка Определите пользователей текущих или конкурирующих продуктов Соберите фокус-группы Создайте архетипы пользователей Воспользуйтесь структурной схемой организации для выявления возможных пользователей

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Слишком много людей вовлечено в выявление требований	Каждая группа хочет быть представлена по политическим причинам Нечетко определены классы пользователей Недостаток представителей пользователей конкретных групп Слишком большое количество классов пользователей	Определите классы пользователей Выявите сторонников или владельцев продукта Определите ответственных за принятие решений по требованиям Отделите политические приоритеты от деловых и технических Ориентируйте проект на удовлетворение нужд привилегированных классов пользователей
Реализованные требования не удовлетворяют нужды пользователей Требования слишком ограничены	Требования содержат ненужные или преждевременные ограничения дизайна Решения представлены как потребности, а требования нужно извлекать из представленных решений Новое ПО должно соответствовать существующим стандартам для приложений и ограничениям пользовательского интерфейса Клиенты не знают, какая информация должна входить в требования Основное внимание при обсуждении требований уделяется дизайну пользовательского интерфейса	Несколько раз спросите «почему?», чтобы понять действительные нужды пользователей, скрывающиеся за представленными требованиями, и обоснование ограничений дизайна Разберитесь с пользовательскими требованиями, прежде чем обсуждать детали пользовательского интерфейса Подготовьте квалифицированных бизнес-аналитиков, которые умеют ставить правильные вопросы и выявлять настоящие потребности Обучите клиентов особенностям создания требований Документируйте бизнес-правила и ограничения

Симптомы	Возможные первопричины	Возможные решения
Отсутствуют необходимые требования	Пользователи не знают, что им нужно Бизнес-аналитик не задал правильные вопросы На выявление требований выделено недостаточно времени Некоторые классы пользователей не представлены Нужные, знающие представители пользователей не участвовали в выявлении требований Участники процесса выявления требований делают неверные предположения Недостаточный обмен информацией между разработчиками и клиентами Пользователи не выражают своих неявных или предполагаемых требований	Подготовьте квалифицированных бизнес-аналитиков, которые умеют задавать правильные вопросы Выявите варианты использования или пользовательские истории Используйте несколько приемов выявления требований Представьте требования в нескольких видах с особым ударением на визуальные модели, стремясь выявить пробелы Проводите рецензирование требований. Используйте множественные инкрементальные процессы рецензирования Проанализируйте требования, используя матрицу CRUD Создавайте прототипы и просите пользователей оценивать их Используйте инкрементальный подход к разработке, чтобы пропущенные требования удалось включить в более поздние версии продукта Создайте и задействуйте матрицу отслеживания связей требований для выявления пропущенных требований

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Указанные требования не верны или неуместны	Вовлечение не тех представителей пользователей или лиц, их заменяющих Представители пользователей выражают свои точки зрения, а не позиции групп, которые они представляют Менеджеры не предоставляют доступ к представителям пользователей Бизнес-требования установлены недостаточно ясно Пользовательские и функциональные требования не соответствуют бизнес-целям	Определите, что именно было неверно в требованиях и почему они были указаны Определите классы пользователей Определите соответствующих сторонников продукта, обучите их и наделите полномочиями Проведите проверку документации требований многофункциональной командой Сообщите о рисках, связанных с неточными требованиями, заинтересованным лицам, обладающим высокими полномочиями Объясните важность качественного представления пользователей высокопоставленным заинтересованным лицам

Проблемы, связанные с анализом

Описанные в следующей таблице симптомы указывают на необходимость более эффективного анализа требований.

Симптомы	Возможные первопричины	Возможные решения
Указаны ненужные требования Во время тестирования обнаруживаются неожиданные функции Функции определяются и реализуются, но не используются	Отсутствие контроля за утверждением требований Разработчики добавляют функции без консультаций с пользователями Пользователи запрашивают сложные решения, а не выражают бизнес-потребности Выявление требований больше ориентируется на функции системы, чем на цели пользователей Разработчики и клиенты поразному трактуют требования Не обнаруживаются связи требований обратно с бизнес-целями	Записывайте источник и обоснование каждого требования Применяйте варианты использования, чтобы сосредоточиться на бизнес-целях пользователей. Выводите функциональные требования на основе анализа вариантов использования или пользовательских историй Определяйте приоритеты требований, чтобы реализовать особенно ценные функции на ранних стадиях проекта Проведите проверку документации требований многофункциональной командой

Симптомы	Возможные первопричины	Возможные решения
Требования недостаточно ясны, чтобы составить тесты	Требования двусмысленные, не полные или недостаточно детальные	Тестировщики должны проверить требования на возможность тестирования и другие проблемы с качеством
Все требования кажутся одинаково важными Все требования имеют первостепенный приоритет Бизнес-аналитики не могут принимать обоснованные решения о компромиссах, когда появляются новые требования	Опасения, что требования с низким приоритетом никогда не будут реализованы Недостаток знаний о бизнесе и его нуждах Информация о ценности и стоимости реализации каждого требования не известна, не доводится или не обсуждается Продукт не пригоден к использованию, пока не реализован большой набор критических функций Неразумные ожидания клиентов или разработчиков Информацию о приоритетах предоставляют только клиенты	Разработайте совместный процесс определения приоритетов требований, чтобы обеспечить баланс ценности для клиента и затратами на реализацию и техническим риском Определяйте приоритеты требований на ранних стадиях Разработайте детальные спецификации требований с высоким приоритетом Используйте инкрементальный подход к разработке или пошаговые версии продукта, чтобы реализовать максимум ценных функций продукта как можно раньше Динамически корректируйте приоритеты требований, остающихся в резерве проекта
Изменяющиеся приоритеты требований	Люди, принимающие решения, не назначены или не уполномочены это делать Внутренняя политическая борьба Неясные бизнес-цели, отсутствие единства мнений относительно бизнес-целей Внешние силы, такие, как распорядительные или юридические факторы Требования и их приоритеты не утверждаются и не согласуются с соответствующими лицами	Документируйте масштабы, цели и приоритеты проекта Обеспечивайте соответствие приоритетов требований бизнес-целям Определите и наделите полномочиями ответственных за принятие решений по требованиям Ведите учет последствий изменений в области расходов, доходов и переноса сроков Используйте итеративный подход к разработке и динамически корректируйте приоритеты требований, остающихся в резерве проекта

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Нет единого мнения относительно приоритетов требований между заинтересованными лицами	У различных классов пользователей противоречивые нужды Недостаток дисциплины, необходимой, чтобы придерживаться первоначальной концепции продукта или концепция меняется в ходе проекта Неясные бизнес-цели, отсутствие единства мнений относительно бизнес-целей Меняющиеся бизнес-цели Неясно, кто принимает решения, связанные с требованиями	Проведите исследование рынка Установите и доведите до всех бизнес-цели Основывайте приоритеты на видении, границах и бизнес-целях Определите предпочтительные классы пользователей или сегменты рынка Определите сторонников продукта для представительства различных классов пользователей Определите и наделите полномочиями ответственных за принятие решений по требованиям
Быстрое сужение границ на поздних стадиях проекта	Нереалистичный оптимизм относительно производительности труда разработчиков Недостаточное начальное и периодическое определение приоритетов Нет доверия приоритетам при определении последовательности реализации и внесения контролируемых изменений границ проекта	Определите приоритеты на ранних стадиях проекта На основе приоритетов решайте, над чем работать сейчас, а что отложить Корректируйте приоритеты, когда вносите новые требования Корректировку границ проекта выполняйте периодически, а не только на поздних стадиях проекта Используйте инкрементальный подход к разработке или поэтапные выпуски, чтобы сохранять ориентацию на пользу для клиента

Симптомы	Возможные первопричины	Возможные решения
Разработчики считают требования расплывчатыми и двусмысленными Разработчикам приходится искать упущенную информацию Разработчики неверно толкуют требования, и им приходится переделывать работу	Аналитики и клиенты не понимают, какой уровень детализации требований нужен разработчикам Клиенты не знают, что им нужно, или не могут ясно это выразить На выявление требований отведено недостаточно времени Бизнес-правила не определены, не доведены или не поняты Формулировки требований содержат много расплывчатых и неоднозначных слов Заинтересованные лица толкуют термины, концепции и определения данных по-разному Клиенты предполагают, что разработчики уже знают достаточно об их области бизнеса и нуждах	Учите аналитиков составлять хорошие требования Избегайте использования субъективных, двусмысленных терминов в специфи-кации На ранних стадиях проекта разработчики и клиенты должны рецензировать требования, чтобы выяснить, достаточно ли те ясны и детализированы Моделируйте требования, чтобы обнаружить пропущенную информацию и добавить деталей Создавайте прототипы и просите пользователей оценивать их Уточняйте требования с возрастающим уровнем детализации Документируйте бизнес-правила Определите значения терминов в словаре Определите элементы данных в словаре данных Поддерживайте эффективное взаимодействие между всеми участниками проекта
Некоторые требования технически неосуществимы	Требования не анализируются должным образом Клиенты не принимают результаты анализа осуществимости Непонимание ограничений новых инструментальных средств и технологий, а также операционной среды	Проведите анализ осуществимости Создавайте экспериментальные образцы-прототипы Привлеките одного из разработчиков поучаствовать в выявлении требований Попросите разработчиков прорецензировать требования на предмет осуществимости Проведите отдельное исследование, исследовательский мини-проект или пилотный проект для оценки осуществимости

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Требования из различных источников или от различных классов пользователей конфликтуют друг с другом Трудно достичь соглашения заинтересованных в проекте лиц относительно требований	Нет единого понимания продукта Не назначены ответственные за принятие решения по требованиях Разные заинтересованные лица понимают бизнес-процессы по-разному Политика играет большую роль в определении требований У различных групп пользователей или сегментов рынка различные нужды, ожидания и бизнес-цели Продукт недостаточно ориентирован на конкретный сегмент рынка Некоторые группы пользователей уже применяют систему, к которой привыкли	Разработайте, утвердите и распространите единый набор бизнес-требований Изучите сегменты целевого рынка и классы пользователей Определите привилегированные классы пользователей для разрешения конфликтов Определите сторонников продукта, чтобы разрешать конфликты внутри каждого класса пользователей Определите и наделите полномочиями ответственных за принятие решений по требованиям Сосредоточьтесь на общих бизнес-интересах, а не защищайте эмоциональные и политические позиции
Требования содержат неясности, информационные пробелы и открытые проблемы	Не назначен ответственный за разрешение неясностей до того, как требования передаются разработчикам Нет времени на разрешение неясностей или проблем до начала реализации	Исследуйте требования для выявления информационных пробелов Назначьте ответственных за разрешение каждой неясности или открытой проблемы Определите приоритеты неясностей, чтобы можно было принимать решения в случае нехватки времени Отслеживайте каждую неясность или открытую проблему до создания базового набора требований
Бизнес-аналитик тратит слишком много времени на анализ требований	Нежелание отдавать работу, пока требования не станут «идеальными» (паралич аналитического процесса) Намерение разработать требование полностью, а не достаточного для дальнейшей работы качества Неправильный выбор приемов анализа в проекте	Сосредоточьте анализ и моделирование на сложных, инновационных или нечетких частей требований Используйте рецензирование для оценки, когда требования достаточно хороши, чтобы можно было продолжать работу с приемлемым уровнем риска

Проблемы, связанные со спецификацией требований

Симптомы, приведенные в следующей таблице, говорят о недостатках процесса спецификации требований в проекте.

Симптомы	Возможные первопричины	Возможные решения
Требования не документированы Требования создают разработчики Клиенты предоставляют подробности требований разработчикам в устной форме Разработчикам приходится выполнять много работы по опытному программированию в попытке выяснить, что хотят клиенты	Никто не знает, что именно нужно создать На выявление и документирование требований выделено мало времени Бытует мнение, что документирование требований замедляет проект Неясно определено, кто отвечает за документацию, или эти люди недостаточно заинтересованы Не определен процесс разработки требований или нет шаблонов Те, кто управляет разработкой, не понимают, не ценят и не ожидают разработки спецификации требований Разработчики полагают, что знают нужды клиентов	Сформулируйте риски, связанные с некачественной спецификацией требований Определите и следуйте процессу разработки требований Определите распределение ролей в команде и получите у конкретных людей согласия на выполнение ролей Проведите обучение членов команды и клиентов в области процессов работы с требованиями Вносите действия, связанные с затратами, ресурсами, задачами и результатами работы над требованиями, в планы и графики работы по проектам Обзаведитесь стандартными шаблонами и хорошими примерами спецификаций требований, которые можно предоставить в общий доступ
Заинтересованные лица предполагают, что функциональность старой системы будет продублирована в новой	Требования для новой системы указаны в виде дельты от плохо документированной существующей системы Бизнес-цели неясны	Проведите обратный инженерный анализ существующей системы, чтобы понять все ее возможности Составьте спецификацию требований, включающую всю желаемую функциональность для новой системы Создайте модель существующих и будущих процессов, чтобы заинтересованные лица четко понимали, что система будет и что не будет делать Не дублируйте старую функциональность, которая может не потребоваться

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Документация требований неточно описывает систему	Изменения, реализованные во время разработки, не вносятся в документацию требований	Следуйте процессу управления изменениями, включающему обновление документации требований при принятии изменений
		Проводите все запросы на изменения через совет по управлению изменениями
		Привлекайте ключевых заинтересованных лиц к рецензированию измененных требований
Существуют различные, противоречащие друг другу версии требований	Плохие приемы управления версиями	Определите и следуйте качественным приемам управления версиями документации требований
	Наличие нескольких «главных» экземпляров документации требований	Храните требования в средстве управления требованиями
	Требования хранятся отдельно в специализированном средстве и в документах, и люди не уверены, какая версия является полномочной	Назначьте менеджера по требованиям ответственным за внесение изменений в требования

Проблемы, связанные с утверждением требований

Трудно быть уверенным, что разработанные вами требования на самом деле достигнут намеченных бизнес-целей. Симптомы этого раздела указывать на недостатки процессов утверждения требований.

Симптомы	Возможные первопричины	Возможные решения
Продукт не достигает бизнес-целей или не отвечает ожиданиям пользователей Клиенты имеют невысказанные, предполагаемые или скрытые требования, которые не были удовлетворены	Клиенты неточно выразили свои нужды Потребности рынка или бизнес-нужды изменились, но отсутствовали механизмы соответствующего пересмотра требований Бизнес-аналитик не задал нужных вопросов Недостаточное вовлечение пользователей в разработку требований Неверно представлены пользователи, участвовавшие в процессе, например заместители реальных пользователей не выражали их действительные нужды Маркетологи неточно оценили нужды рынка, особенно когда речь идет об инновационных продуктах с неопределенными требованиями Участники проекта сделали неверные предположения	Проведите исследование рынка, чтобы понять сегменты рынка и их потребности Привлекайте сторонников продукта, представляющих каждый привилегированный класс пользователей, на протяжении всего проекта Научите бизнес-аналитиков ставить правильные вопросы. Разработайте варианты использования, которые обеспечат понимание бизнес-задач Привлеките клиентов к участию в рецензировании требований Создавайте прототипы и просите пользователей оценивать их Попросите пользователей создать приемочные тесты и критерии приемки Установите эффективные механизмы работы с изменениями, чтобы требования менялись вместе с изменением бизнес-реалий
Продукт не достигает целевой производительности или не удовлетворяет другим ожиданиям пользователей по качеству	Требованиям к атрибутам качества не обсуждались во время выявления требований Заинтересованные в проекте лица не понимают нефункциональных требований и их важности В используемом шаблоне спецификации требований или специализированном средстве не предусмотрено разделов для нефункциональных требований Пользователи не выражают своих предположений о качественных характеристиках системы Атрибуты качества не были точно определены, чтобы все заинтересованные в проекте лица понимали их одинаково	Расскажите бизнес-аналитикам и клиентам о нефункциональных требованиях и о том, как их определять Добейтесь, чтобы в процессе выявления требований бизнес-аналитики учитывали нефункциональные требования Используйте шаблон спецификации требований, содержащий разделы для нефункциональных требований Используйте Planguage для точного указания атрибутов качества

(*см. след. стр.*)

Проблемы, связанные с управлением требованиями

Один из признаков недостаточно качественного управления требованиями заключается в реализации не всех запланированных требований.

Симптомы	Возможные первопричины	Возможные решения
Некоторые запланированные требования не были реализованы	Спецификация требований некачественно организована или написана Требования не были по отдельности определены и маркированы Разработчики не следовали спецификации требований к ПО Спецификация требований не была донесена до всех разработчиков Об изменениях не сообщалось всем, кого это затрагивало Требования непреднамеренно пропускались во время реализации Обязанности по реализации требований не были детально расписаны Состояние требований по отдельности отслеживалось неточно	Своевременно обновляйте спецификацию требований и обеспечивайте доступ к ней всей команде Позаботьтесь, чтобы в процесс управления изменениями входил обмен информацией с заинтересованными лицами Храните требования в средстве управления требованиями Отслеживайте состояние отдельных требований Создайте и применяйте матрицу отслеживания связей требований Четко определите обязанности по реализации ПО Учите аналитиков составлять ясные и лаконичные требования

Проблемы, связанные с управлением изменениями

Существует много признаков того, что в проекте ПО неправильно работают с запросами на изменения, и некоторые из них перечислены в следующей таблице.

Симптомы	Возможные первопричины	Возможные решения
Требования часто изменяются Многие изменения вносятся в требования на поздних стадиях процесса разработки Изменения приводят к нарушению условий поставки продукта	Пользователи не знают, что им нужно Бизнес-процессы или требования рынка меняются Не все нужные люди участвовали в выявлении и утверждении требований Требования с самого начала недостаточно хорошо определены Базовая версия требований не была определена или не согласована Внешние факторы, например государственные и регулирующие органы диктуют изменения Исходные требования содержали много готовых решений, не удовлетворявших реальных нужд Рыночные нужды были не очень хорошо поняты	Усовершенствуйте приемы выявления требований Введите и исполняйте процесс управления изменениями Создайте совет по управлению изменениями для принятия решений по предлагаемым изменениям Проводите анализ последствий изменений до их принятия Заинтересованные в проекте лица должны рецензировать требования до их включения в базовую версию Проектируйте ПО в расчете на широкие возможности для внесения и принятия изменений Включите в график проекта резервы на случай непредвиденных обстоятельств, чтобы иметь возможность вносить некоторые изменения. Используйте приемы инкрементального моделирования, чтобы быстро реагировать на изменения требований Старайтесь жестко следовать графику, а при необходимости согласовывайте выпуск продукта с меньшим набором функций с расчетом на реализацию остальных функций в следующем выпуске

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Часто добавляются новые требования Расширение границ проекта приводит к нарушению условий поставки продукта	Выявление требований выполнено неполно Недостаточное вовлечение пользователей в разработку требований Бизнес-потребности или среда быстро меняются Предметная область была понята неправильно Заинтересованные в проекте лица не понимают или не соблюдают границы проекта Руководство, отдел маркетинга или клиенты требуют новых функций без учета последствий этого для проекта	Усовершенствуйте приемы выявления требований Определите и доведите до остальных границы проекта Нужные люди должны принимать точные бизнес-решения об изменении границ проекта Проведите анализ первопричин того, откуда и почему появляются новые требования Проводите анализ последствий изменений до их принятия Обеспечьте вклад всех классов пользователей Включите в график проекта резервы на непредвиденные обстоятельства, чтобы иметь возможность вносить некоторые дополнения Используйте приемы инкрементальной разработки, чтобы быстро реагировать на новые требования
Требования входят и выходят за границы проекта	Концепция и границы проекта определены нечетко Бизнес-цели определены или доводятся недостаточно ясно Границы проекта постоянно изменяются, возможно в ответ на изменения потребностей рынка Приоритеты требований плохо определены Ответственные за принятие решений не согласны с границами проекта	Ясно определите бизнес-цели, концепцию и границы проекта Используйте положение о границах проекта для принятия решения о том, входят ли предлагаемые изменения в его границы Записывайте обоснование отклонения предложенных требований Проследите, чтобы совет по управлению изменениями имел должный состав и выработал единое мнение о границах проекта Используйте методы инкрементальной разработки, чтобы гибко учитывать изменяющиеся границы проекта Сосредоточьтесь на реализации стабильных требований

Симптомы	Возможные первопричины	Возможные решения
Определение границ проекта изменяется после начала разработки	Плохо определенные, плохо понятые или постоянно меняющиеся бизнес-цели	Определите бизнес-цели и приведите концепцию и границы проекта в соответствие с ними
	Плохое понимание сегментов и потребностей рынка	Определите заинтересованных лиц, принимающих решения на уровне бизнес-требований
	Выпуск конкурирующих продуктов	Лица, принимающие решения, должны проверить документ концепции и границ
	Ключевые заинтересованные в проекте лица не прорецензировали и не утвердили требования	Применяйте процесс управления изменениями для внедрения изменений
	Состав ключевых заинтересованных лиц меняется в ходе проекта	Заново согласовывайте сроки и ресурсы при изменении направления развития проекта
Об изменениях в требованиях не сообщается всем заинтересованным лицам	Изменения требований не доводятся до всех, кого они касаются	Для каждого требования назначьте ответственного
	Спецификации требований не обновляются при изменении требований	Определите связи между требованиями и другими артефактами
	Клиенты запрашивают изменения непосредственно у разработчиков	Включайте в обмен информацией по требованиям все области, которые она затрагивает
	Не всем обеспечен легкий доступ к документации требований	Установите процесс управления изменениями, включающий механизмы обмена информацией
	Неформальные пути передачи информации исключают из процесса некоторых участников проекта	Все изменения требований выполняйте в рамках процесса управления изменениями
	Неясно, кого нужно информировать об изменениях	Используйте инструментальное средство управления требованиями для обеспечения доступа заинтересованных лиц
	Отсутствует процесс управления изменениями	Усовершенствуйте совместную работу и обмен информацией среди участников проекта и другими заинтересованными лицами
	Непонятны взаимосвязи между требованиями	

(*см. след. стр.*)

Симптомы	Возможные первопричины	Возможные решения
Предложения об изменениях в требованиях теряются Состояние запросов на изменения неизвестно	Неэффективный или неопределенный процесс управления изменениями Процесс управления изменениями не соблюдается	Примите практичный, эффективный процесс управления изменениями и научите заинтересованных лиц следовать ему Распределите обязанности по выполнению этапов процесса управления изменениями Позаботьтесь, чтобы процесс управления изменениями выполнялся Используйте инструментальные средства управления требованиями для учета изменений и состояния каждого требования
Заинтересованные лица обходят процесс управления изменениями Клиенты запрашивают изменения непосредственно у разработчиков	Процесс управления изменениями непрактичен и неэффективен Совет по управлению изменениями работает неэффективно Заинтересованные в проекте лица не понимают или не принимают процесс управления изменениями Руководство не требует соблюдения процесса управления изменениями	Позаботьтесь, чтобы процесс управления изменениями был практичным, эффективным, действенным, понятным и доступным для всех заинтересованных лиц Обеспечьте гибкость процесса управления изменениями, чтобы он одинаково эффективно справлялся как с маленькими, так и большими изменениями Создайте работоспособный совет по управлению изменениями Заручитесь поддержкой руководства в исполнении и лидерстве в процессе управления изменениями Внедрите политику, согласно которой изменения требований выполняются только в рамках процесса изменения требований

Симптомы	Возможные первопричины	Возможные решения
Изменения в требованиях требуют гораздо больше усилий, чем планировалось Изменения затрагивают больше компонентов системы, чем ожидалось Изменения противоречат другим требованиям Изменения снижают качество системы	Недостаточный анализ последствий предлагаемых изменений в требованиях	Утвердите процедуру и контрольные списки анализа последствий изменений
	Разработчики недооценивают влияние изменений в требованиях	Внедрите анализ последствий в процесс управления изменениями
	Решения по изменениям принимаются не уполномоченными на то сотрудниками	Используйте информацию отслеживания связей для оценки последствий предлагаемых изменений
	Члены команды боятся честно оценивать последствия предлагаемых изменений	Сообщайте об изменениях всем заинтересованным в проекте лицам, которых это затрагивает
	Запросы на изменения не содержат достаточно информации для качественного выполнения анализа влияния изменения	По мере необходимости заново согласовывайте обязательства и принимайте необходимые компромиссы, когда предлагаются изменения

Приложение B

Примеры документации требований

В данном приложении на примере небольшого гипотетического проекта под названием Cafeteria Ordering System (COS) проиллюстрированы некоторые описанные в этой книге документы и диаграммы, необходимые при разработке требований. К ним относятся:

- документ о концепции и границах проекта;
- документ о концепции и границах проекта;
- часть спецификации требований к ПО;
- несколько фрагментов моделей анализа, в том числе дерево функций, контекстная диаграмма, диаграмма «сущность–связь» и диаграмма переходов состояний;
- фрагмент словаря данных;
- несколько бизнес-правил.

Поскольку это лишь пример, эти элементы документации требований намеренно не приводятся в законченном виде. Я хочу здесь просто показать в общих чертах, как различные типы информации, необходимой при создании требований, относятся друг к другу и как можно записывать содержание каждого раздела. Информацию в этих примерах можно организовывать и группировать по-разному в зависимости от задачи — объединять в единый документ в маленьком проекте или хранить в средстве управления требованиями. Ясность, полнота и простота использования документации требований — важнейшие задачи при ее составлении. Документы обычно строятся по шаблонам, описанным в предыдущих главах, но здесь из-за малых масштабов этого проекта некоторые разделы шаблона объединены. Для каждого проекта необходимо учитывать, как адаптировать стандартные шаблоны организации, чтобы они лучше соответствовали размеру и природе проекта.

Документ о концепции и границах проекта

1. Бизнес-требования

1.1. Исходные данные

Большинство сотрудников Process Impact в настоящее время тратят в среднем 65 минут в день, чтобы выбрать, оплатить и съесть обед в кафетерии. Около 20 минут уходит на то, чтобы дойти до кафетерия и вернуться на рабочее место, выбрать еду и оплатить ее наличными или с помощью кредитной карты. Таким образом, сотрудники проводят около 90 минут вне рабочих мест. Некоторые звонят в кафетерий заранее, чтобы блюда были готовы к их приходу. Они не всегда получают то, что заказали, потому что некоторые блюда заканчиваются до их прихода. Кафетерий впустую расходует значительный объем продуктов, которые не реализуются, и их приходится выбрасывать. Те же проблемы возникают утром и вечером, хотя гораздо меньше сотрудников завтракает и ужинает, чем обедает.

1.2. Возможности бизнеса

Многие сотрудники попросили создать систему, которая позволила бы посетителям кафетерия делать заказ (определенный как набор из одного или большего числа блюд из меню) по сети, чтобы его можно было забрать или заказать доставку в назначенное место в офисе компании в указанный день в указанное время. Подобная система сэкономит значительное время и позволит получать те блюда, которые хотят клиенты. Это улучшит как качество рабочей среды, так и производительность труда. Кроме того, предоставленные заранее сведения о том, какие блюда хотят получить клиенты, позволит уменьшить потери и увеличит эффективность работы персонала кафетерия. Если сотрудники получат в будущем возможность заказывать доставку блюд из близлежащих ресторанов, то у них появится больший выбор при меньшей цене, так как фирма сможет заключать оптовые соглашения с ресторанами.

1.3. Бизнес-цели

BO-1 Уменьшить потери продуктов в кафетерии на 40% в течение 6 месяцев после первого выпуска системы *[Этот пример показывает использование Planguage для точной формулировки бизнес-цели.]*

Масштабы: стоимость продуктов, выбрасываемых каждую неделю персоналом кафетерия.

Способ измерения: исследование системы учета запасов кафетерия.

Показатели в прошлом: 30% (2013 г., первоначальное исследование).

Планируемые показатели: менее 20%.

Обязательные показатели: менее 20%.

BO-2 Снизить эксплуатационные расходы кафетерия на 15% в течение 12 месяцев после первого выпуска системы.
BO-3 Увеличить среднее эффективное рабочее время каждого сотрудника на 15 минут в день в течение 6 месяцев после первого выпуска системы.

1.4. Критерии успеха

SM-1 75% сотрудников, которые пользовались кафетерием как минимум три раза в неделю в третьем квартале 2013 года, должны начать использовать Cafeteria Ordering System как минимум раз в неделю в течение 6 месяцев после первого выпуска системы.
SM-2 Достичь увеличения среднего рейтинга по ежеквартальному опросу об удовлетворенности работой кафетерия на 0,5 балла по сравнению с третьим кварталом 2013 года по шкале от 1 до 6 в течение 3 месяцев после первого выпуска системы и на 1,0 балла в течение 12 месяцев.

1.5. Видение решения

Для сотрудников, желающих заказывать еду в кафетерии компании или в местных ресторанах через Интернет, Cafeteria Ordering System — это интернет-приложение или приложение для смартфона, которое принимает индивидуальные или групповые заявки на питание, взимает оплату и инициирует доставку готовых блюд к указанному пункту на территории Process Impact. В отличие от имеющихся в настоящее время служб заказа по телефону и вручную, сотрудникам, использующим Cafeteria Ordering System, не придется приходить в кафетерий, чтобы получать заказанные блюда, что сэкономит им время, кроме того, увеличится ассортимент доступных им блюд.

1.6. Бизнес-риски

RI-1 Профсоюз работников кафетериев может потребовать пересмотра контрактов сотрудников кафетерия, чтобы они отражали новое распределение обязанностей и график работы кафетерия. (Вероятность = 0,6; ущерб = 3.)
RI-2 Слишком мало сотрудников могут сразу принять новую систему, что уменьшит прибыль от инвестиций в разработку системы и изменений в схеме работы кафетерия. (Вероятность = 0,3; ущерб = 9.)
RI-3 Близлежащие рестораны могут не согласиться предоставить скидки, что уменьшит удовлетворенность сотрудников системой и возможно, ее использование. (Вероятность = 0,3; ущерб = 3.)
RI-4 Имеющихся возможностей может оказаться недостаточно, из-за чего сотрудники смогут не всегда получать свои заказы и заказывать доставку на нужное время. (Вероятность = 0,5; ущерб = 6.)

1.7. Предположения и зависимости

AS-1 У работников кафетерия будут системы с соответствующими интерфейсами для обработки ожидаемого числа заказываемых блюд.

AS-2 Число работников кафетерия и автомобилей будет таким, что все блюда будут доставляться в течение 15 минут в рамках указанного времени.
DE-1 Если в ресторане есть собственная система заказов, Cafeteria Ordering System должна поддерживать двустороннюю связь с ней.

2. Рамки и ограничения проекта

2.1. Основные функции

FE-1 Заказ и оплата блюд из меню кафетерия для получения в кафетерии или с доставкой.
FE-2 Заказ и оплата блюд с доставкой из близлежащих ресторанов.
FE-3 Создание, просмотр, изменение и удаление одинарной или регулярной заявок на питание или на ежедневные специальные блюда.
FE-4 Создание, просмотр, изменение и удаление меню кафетерия.
FE-5 Просмотр списка ингредиентов и сведения о питательности блюд в меню кафетерия.
FE-6 Обеспечение доступа к системе через корпоративную интрасеть, смартфон, планшет или через внешнее подключение к Интернету для авторизованных сотрудников.

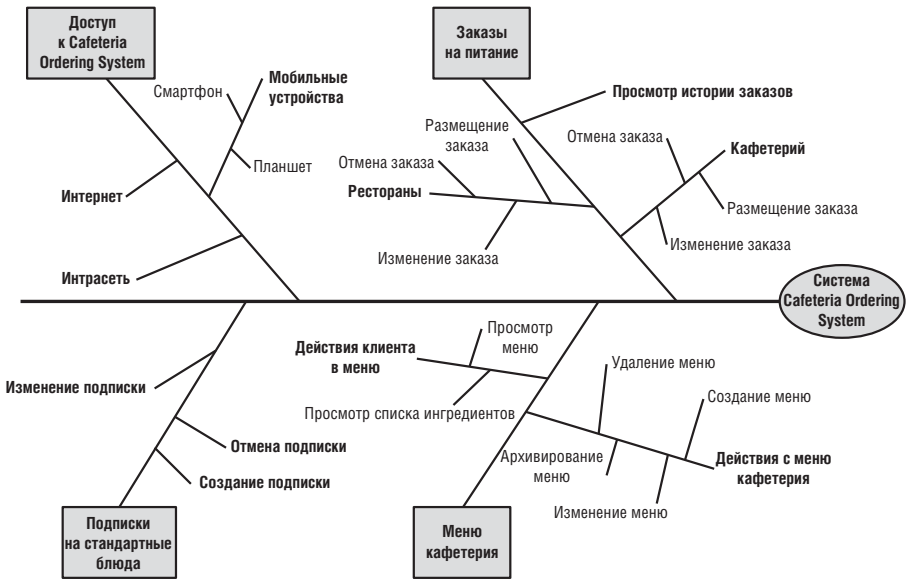


Рис. B-1. Частичное дерево функций системы Cafeteria Ordering System

2.2. Состав первого и последующих выпусков системы

Функция	Выпуск 1	Выпуск 2	Выпуск 3
FE-1. Заказ в кафетерии	Только стандартные функции из меню обедов; оплата заказов производится только посредством удержания из зарплаты	Прием платежа кредитной или дебетовой картой	Прием заказов на завтрак и ужин
FE-2. Заказы из ресторанов	Не реализована	Блюда доставляются только на территории компании	Реализована полностью
FE-3. Подписки на стандартные блюда	Не реализована	Реализация, если позволит время	Реализована полностью
FE-4. Меню	Создание и просмотр меню	Модификация, удаление и архивирование меню	
FE-5. Список ингредиентов	Не реализована	Реализована полностью	
FE-6. Доступ к системе	Интрасеть и доступ через Интернет извне	Приложения для телефонов и планшетов с iOS и Android	Приложения для телефонов и планшетов с Windows Phone

2.3. Ограничения и исключения

LI-1 На некоторые пункты меню кафетерия доставка не распространяется, поэтому блюда, доступные клиентам Cafeteria Ordering System, будут подмножеством полных меню кафетерия.

LI-2 Cafeteria Ordering System применяется только для кафетерия главного офиса Process Impact в г. Клакамас, штат Орегон.

3. Бизнес-контекст

3.1. Профили заинтересованных лиц

Заинтересованное лицо	Основная ценность	Отношение	Основные интересы	Ограничения
Руководство компании	Увеличение производительности труда сотрудников; сокращение затрат в кафетерии	Сильная поддержка вплоть до выпуска 2; поддержка выпуска 3 в зависимости от результатов предыдущих выпусков	Экономия расходов должна превысить затраты на разработку и использование	Не определены
Сотрудники кафетерия	Более эффективное использование рабочего времени сотрудников в течение дня; большее удовлетворение клиентов	Озабоченность взаимоотношениями с профсоюзом и возможным сокращением персонала; в остальном — все воспринимается нормально	Сохранение рабочих мест	Необходимость обучения сотрудников работе с Интернетом; необходимость в персонале и транспорте для доставки
Постоянные клиенты кафетерия	Лучший выбор блюд; экономия времени; удобство	Большой энтузиазм, но могут использовать систему меньше, чем ожидается, из-за социальной значимости обедов в кафетерии и ресторанах	Простота использования; надежность доставки; возможность выбора блюд	Необходимость доступа к корпоративной интрасети, к Интернету или требуется мобильное устройство
Отдел расчета зарплаты	Отсутствие какой-либо выгоды; необходимость создания схемы удержания стоимости заказов из зарплаты	Не особо счастливы относительно предстоящей работы над ПО, но понимают ценность для компании и сотрудников	Минимум изменений в текущих приложениях расчета зарплаты	Еще не выделено никаких ресурсов на изменение ПО

(*см. след. стр.*)

Заинтересованное лицо	Основная ценность	Отношение	Основные интересы	Ограничения
Менеджеры ресторанов	Увеличение продаж; выход на новые области рынка для привлечения новых клиентов	Поддерживают, но с осторожностью	Минимум новых технологий; озабоченность ресурсами и затратами, необходимыми для доставки блюд	Могут не иметь персонала и возможностей для обработки нужных объемов заказов; не у всех меню представлены в Интернете

3.2. Приоритеты проекта

Область	Ограничения	Движущая сила	Степень свободы
Функции	Все функции, запланированные на выпуск 1.0, должны быть полностью реализованы		
Качество	95% пользовательских проверочных тестов должны быть выполнены; все тесты на защищенность должны быть выполнены		
Сроки			По плану выпуск 1 должен быть доступен к концу I квартала следующего года, выпуск 2 — к концу II квартала, допустима задержка до 2 недель без пересмотра сроков куратором проекта
Расходы			До 15% перерасхода по бюджету возможны без пересмотра куратором проекта
Персонал		Планируемый состав команды: работающий на полставки менеджер проекта, 2 разработчика,тестировщик, работающий на полставки; при необходимости могут быть дополнительно привлечены разработчик и тестировщик, работающие на полставки	

3.3. Особенности развертывания

ПО веб-сервера нужно обновить до последней версии. В рамках второго выпуска нужно разработать приложения для смартфорнов и планшетов под управлением iOS и Android, а в третьем выпуске нужно выпустить приложения для смартфонов и планшетов с Windows Phone. К моменту готовности второго выпуска все соответствующие изменения должны быть выполнены. Нужно разработать видеоролики длительностью не более пяти минут, обучающие пользователей работе с интернет-версией и приложениями системы Cafeteria Ordering System.

Варианты использования

Различные классы пользователей определили следующие варианты использования и основных действующих лиц для Cafeteria Ordering System.

Основное действующее лицо	Вариант использования
Клиент	Заказ блюд
	Изменение заказа
	Отмена заказа
	Просмотр меню
	Регистрация для оплаты посредством удержания из зарплаты
	Отмена регистрации для оплаты посредством удержания из зарплаты
	Изменение подписки
Менеджер меню	Создание меню
	Изменение меню
	Удаление меню
	Архивирование меню
	Определение блюд на заказ
Сотрудники кафетерия	Приготовление блюд
	Генерация запроса на оплату
	Запрос на доставку
	Генерация отчетов по использованию системы
Курьер	Регистрация доставки блюд
	Распечатка инструкций по доставке

(*см. след. стр.*)

Идентификатор и название варианта использования	**UC-1. Заказ блюд**		
Автор	Притхви Радж	Дата создания:	04.10.2013
Основное действующее лицо	Клиент	Дополнительные действующие лица:	Система Cafeteria Ordering System
Описание	Клиент обращается в Cafeteria Ordering System из корпоративной интрасети или через Интернет, просматривает меню на определенную дату, выбирает блюда и делает заказ на получение блюд в кафетерии или доставку в определенный пункт в пределах определенного 15-минутного промежутка времени		
Условие-триггер	Клиент выражает намерение заказать блюдо		
Предварительные условия	PRE-1. Клиент вошел в систему Cafeteria Ordering System. PRE-2. Клиент зарегистрировался для оплаты посредством удержания из зарплаты		
Выходные условия	POST-1. Заказ на доставку блюд сохранен в Cafeteria Ordering System с состоянием «Принят» POST-2. Список доступных блюд обновлен с учетом элементов этого заказа POST-3. Остающиеся резервы возможности доставки в указанный интервал времени обновлены с учетом этого заказа		
Нормальное направление	**1.0 Заказ одного набора блюд** 1. Клиент запрашивает просмотр меню за указанную дату (см. 1.0. Исключение-1, 1.0. Исключение-2) 2. Cafeteria Ordering System отображает меню доступных блюд и ежедневных специальных блюд 3. Клиент выбирает одно или более блюд из меню. (см. 1.1) 4. Клиент указывает, что заказ блюд завершен. (см. 1.2) 5. Cafeteria Ordering System отображает заказанные блюда из меню, стоимость каждого из них и общую сумму, включая все налоги и стоимость доставки 6. Клиент подтверждает заказ блюд (продолжение нормального направления) или делает запрос на изменение заказа (обратно к п. 2) 7. Система выводит доступные периоды времени доставки на дату доставки 8. Клиент выбирает время доставки и указывает пункт доставки 9. Клиент указывает метод оплаты 10. Система подтверждает, что заказ принят 11. Система отправляет клиенту сообщение электронной почты с подтверждением деталей заказа, цены и указаниями по доставке 12. Система сохраняет заказ в базе данных, посылает информацию о заказанных блюдах в систему учета запасов кафетерия и обновляет доступные периоды времени доставки		

Идентификатор и название варианта использования	UC-1. Заказ блюд
Альтернативные направления	**1.1 Заказ нескольких идентичных блюд** 1. Клиент делает запрос на заказ определенного числа идентичных блюд. (см. 1.1.E1) 2. Возврат к п.4 нормального направления **1.2 Заказ нескольких блюд** 1. Клиент заказывает еще одно блюдо 2. Возврат к п.1 нормального направления
Исключения	**1.0.E1 Текущее время — после истечения крайнего срока заказов** 1. Система извещает клиента, что уже слишком поздно делать заказ на сегодня 2а. Если клиент отменяет ввод заказа, система завершает вариант использования 2б. В противном случае клиент запрашивает другую дату и система начинает вариант использования сначала **1.0.E2 Не осталось резервов времени доставки** 1. Система сообщает клиенту, что нет незанятого времени доставки на выбранное число 2а. Если клиент отменяет ввод заказа, система завершает вариант использования 2б. В противном случае клиент делает запрос, чтобы самому получить заказ в кафетерии, и продолжается нормальное направление с пропуском пунктов 7 и 8 **1.1.E1 Невозможно выполнить заказ на указанное количество одинаковых блюд** 1. Система извещает клиента о максимальном числе одинаковых блюд, заказ на которое она способна принять 2а. Клиент изменяет количество заказов на одинаковые блюда, и возвращается к п.4 нормального направления 2б. В противном случае клиент отменяет ввод заказа, а система завершает вариант использования
Приоритет	Высокий
Частота использования	Приблизительно 300 пользователей, в среднем по одному обращению в день Пиковая нагрузка этого варианта использования приходится на период с 9:00 до 10:00 местного времени
Бизнес-правила	BR-1, BR-2, BR-3, BR-4, BR-11, BR-12, BR-33

(*см. след. стр.*)

Идентификатор и название варианта использования	**UC-1. Заказ блюд**
Другая информация:	1. Клиент должен иметь возможность отменить заказ в любой момент времени до подтверждения заказа 2. Клиент должен иметь возможность просматривать все заказы за последние шесть месяцев и повторить один из них в качестве нового заказа при условии, что все его пункты присутствуют в меню на указанную дату доставки. (приоритет — средний) [***Примечание:*** *Это также можно показать как альтернативное направление этого варианта использования*] 3. По умолчанию используется текущая дата при условии, что клиент использует систему до истечения крайнего срока заказов
Предположения:	Предполагается, что 15% клиентов будут заказывать спецпредложение дня (источник: данные кафетерия за предыдущие шесть месяцев)

[***Примечание:*** *Следующий вариант использования описан менее детально, чем UC-1, чтобы показать, что не всегда нужно полностью указывать все подробности варианта использования, если у разработчиков уже есть нужная информация из другого источника*]

Идентификатор и название варианта использования	**UC-5. Регистрация на оплату через удержание из зарплаты**		
Автор	Ненси Андерсон	Дата создания:	15.09.13
Основное действующее лицо	Клиент	Дополнительные действующие лица	Система расчета зарплаты
Описание	Клиенты кафетерия, использующие Cafeteria Ordering System и заказывающие блюда с доставкой, должны быть зарегистрированы для оплаты посредством удержания из зарплаты. Для безналичных покупок через Cafeteria Ordering System кафетерий будет выставлять счета на оплату в системе расчета зарплат, которая будет удерживать стоимость заказов из той суммы, которую клиент должен получить в следующий раз прямым зачислением в депозит в день зарплаты		
Условие-триггер	Клиент запросил регистрацию для оплаты посредством удержания из зарплаты или клиент согласился на регистрацию, когда его об этом спросила система Cafeteria Ordering System		
Предварительные условия	PRE-1. Клиент вошел в систему Cafeteria Ordering System		

Идентификатор и название варианта использования	UC-5. Регистрация на оплату через удержание из зарплаты
Выходные условия	POST-1. Клиент зарегистрирован для оплаты посредством удержания из зарплаты
Нормальное направление	**5.0 Регистрация для оплаты посредством удержания из зарплаты** 1. Клиент запрашивает в системе расчета зарплаты информацию, может ли он зарегистрироваться для удержания из зарплаты 2. Система расчета зарплат подтверждает, что клиенту предоставлено это право на удержания из зарплаты 3. Система просит клиента подтвердить желание зарегистрироваться для оплаты посредством удержания из зарплаты 4. В случае положительного ответа Cafeteria Ordering System отправляет запрос системе расчета зарплат на включение оплаты посредством удержания из зарплаты для клиента 5. Система расчета зарплат подтверждает, что оплата посредством удержания из зарплаты включена 6. Cafeteria Ordering System уведомляет клиента, что оплата посредством удержания из зарплаты включена
Альтернативные направления	Нет
Исключения.	5.0.E1. Клиент не имеет права на оплату посредством удержания из зарплаты 5.0.E2. Клиент уже зарегистрирован для оплаты посредством удержания из зарплаты
Приоритет	Высокий
Бизнес-правила	BR-86 и BR-88 управляют правом клиента на оплату посредством удержания из зарплаты
Другая информация	Нужно ожидать высокой частоты исполнения этого варианта использования в первые две недели после выпуска системы

[***Примечание:*** *Следующий вариант использования описан очень сжато, чтобы показать, что не всегда нужно полностью заполнять весь шаблон варианта использования, если у разработчиков уже есть нужная информация из другого источника. Рекомендуется планировать заранее, какие варианты использования требуют детализации, а какие нет.*]

Идентификатор и название варианта использования	UC-9. Изменение меню		
Автор	Марк Хассалл	Дата создания:	07.10.13
Описание:	Менеджер меню кафетерия должен иметь возможность открывать меню на определенную дату в будущем, чтобы добавить новые или удалить или изменить уже имеющиеся в меню блюда, создать и изменить специальные блюда или скорректировать цены, а затем сохранить измененное меню		
Исключения	Нет меню на выбранную дату; нужно отобразить сообщение об ошибке и дать менеджеру меню возможность ввести новую дату		
Приоритет	Высокий		
Бизнес-правила	BR-24		
Другая информация	Некоторые блюда не пригодны для доставки, и поэтому меню для клиентов Cafeteria Ordering System, заказывающих блюда с доставкой, не всегда в точности соответствует меню в кафетерии. У менеджера меню должна быть возможность отметить такие блюда как непригодные для доставки		

Спецификация требований к ПО

1. Введение

1.1. Назначение

Эта спецификация требований к ПО описывает функциональные и нефункциональные требования к выпуску 1.0 Cafeteria Ordering System (COS). Этот документ предназначен для команды, которая будет реализовывать и проверять корректность работы системы. Кроме специально обозначенных случаев, все указанные здесь требования имеют высокий приоритет и приписаны к выпуску 1.0.

1.2. Соглашения, принятые в документах

В этой спецификации нет никаких типографских условных обозначений.

1.3. Границы проекта

Cafeteria Ordering System позволит сотрудникам Process Impact заказывать блюда в кафетерии компании через Интернет для доставки в указанные пункты на территории компании. Детальное описание продукта приведено в документе «*Cafeteria Ordering System Vision and Scope Document*» [1], где перечислены функции, полная или частичная реализация которых запланирована в этом выпуске.

1.4. Ссылки

1. Wiegers, Karl. Cafeteria Ordering System Vision and Scope Document, *www.processimpact.com/projects/COS/COS Vision and Scope.docx*
2. Beatty, Joy. Process Impact Intranet Development Standard, Version 1.3, *www.processimpact.com/corporate/standards/PI Intranet Development Standard.pdf*
3. Rath, Andrew. Process Impact Internet Application User Interface Standard, Version 2.0, *www.processimpact.com/corporate/standards/PI Internet UI Standard.pdf*

2. Общее описание

2.1. Общий взгляд на продукт

Cafeteria Ordering System — это новая система, которая заменяет текущие ручные процессы заказа и получения обедов в кафетерии Process Impact. Контекстная диаграмма на рис. B-1 показывает внешние объекты и системные интерфейсы для версии 1.0. Предполагается выпустить несколько версий системы, чтобы в конечном итоге удалось встроить ее в службу заказов нескольких близлежащих ресторанов, работающую через Интернет, а также в службы авторизации кредитных и дебетовых карт.

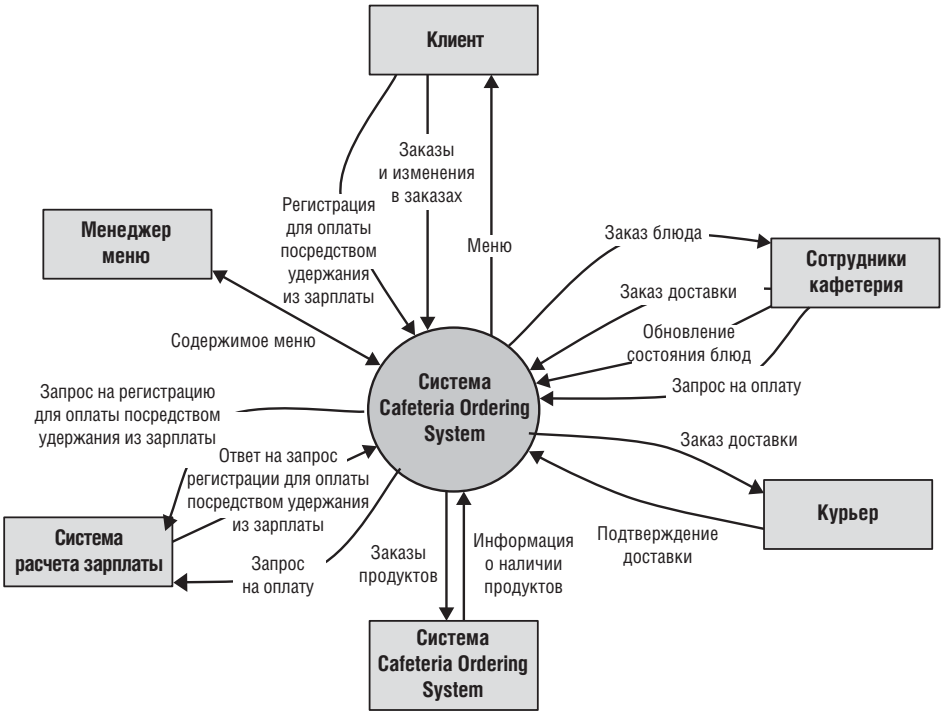


Рис. B-2. Контекстная диаграмма для выпуска 1.0 системы Cafeteria Ordering System

2.2. Классы и характеристики пользователей

Класс пользователей	Описание
Клиент (привилегированный)	Клиент — это сотрудник Process Impact, желающий заказывать питание с доставкой из кафетерия компании. Всего потенциальных клиентов — 600, из которых 400, как ожидается, будут использовать Cafeteria Ordering System в среднем 5 раз в неделю. Иногда клиенты будут заказывать питание на нескольких человек (мероприятия или гости). Ожидается, что 60% заказов будут поступать через корпоративную интрасеть, а 40% — с домашних компьютеров или с применением приложений для смартфонов или планшетов
Сотрудники кафетерия	В кафетерии Process Impact в настоящее время работает около 20 сотрудников, которые будут получать заказы через Cafeteria Ordering System, готовить блюда, упаковывать их для доставки, печатать инструкции по доставке и запрашивать доставку. Большинство сотрудников кафетерия придется обучать работе с компьютером и использованию Cafeteria Ordering System
Менеджер меню	Менеджер меню — это сотрудник кафетерия, отвечающий за создание и поддержку меню на каждый день, в котором указано, какие блюда имеются в наличии в кафетерии. Некоторые блюда в меню могут быть недоступны для доставки. Менеджер меню также определяет спецпредложение дня кафетерия. Менеджер меню должен периодически редактировать меню
Курьер	Готовя заказы к доставке, сотрудники кафетерия будут отравлять запросы на доставку на смартфон курьера. Курьер будет забирать заказ и доставлять их клиентам. Главное взаимодействие сотрудника по доставке с системой будет заключаться в подтверждении успеха (или неудачи) доставки

2.3. Операционная среда

OE-1 Система Cafeteria Ordering System работает со следующими браузерами: Windows Internet Explorer версии 7, 8 и 9, Firefox версии с 12 по 26, Google Chrome (все версии) и Apple Safari версии с 4.0 по 8.0.

OE-2 Система Cafeteria Ordering System установлена на сервере, работающем под управлением текущих утвержденных корпорацией версий Red Hat Linux и Apache HTTP Server.

OE-3 Cafeteria Ordering System должна допускать доступ пользователей через корпоративную интрасеть, VPN-канал и со смартфонов и планшетов под управлением Android, iOS и Windows.

2.4. Ограничения дизайна и реализации

CO-1 Документация системы по дизайну, коду и сопровождению должна соответствовать *Process Impact Intranet Development Standard, версия 1.3* [2].

CO-2 Система должна использовать текущую версию СУБД Oracle, являющуюся корпоративным стандартом.
CO-3 Весь код HTML должен соответствовать стандарту HTML 5.0.

2.5. Предположения и зависимости

AS-1 Кафетерий открыт для завтраков, обедов и ужинов каждый рабочий день компании, когда предполагается присутствие сотрудников на рабочих местах.
DE-1 Работа Cafeteria Ordering System зависит от изменений в системе расчета зарплат, позволяющих принимать запросы на оплату за питание, заказанное через COS.
DE-2 Работа Cafeteria Ordering System зависит от изменений в системе учета запасов кафетерия, позволяющих обновлять информацию о наличии блюд по мере принятия заказов Cafeteria Ordering System.

3. Системные функции

3.1. Заказ блюд из кафетерия

3.1.1. Описание

Клиент кафетерия, личность которого подтверждена, может заказывать набор блюд либо с доставкой в указанное место на территории компании, либо для получения его в кафетерии. Клиент может отменить или изменить заказ, если блюда еще не приготовлены. Приоритет — высокий.

3.1.2. Функциональные требования

Заказ.Размещение:	**Размещение заказа блюд**
.Регистрация:	Система должна подтвердить, что клиент зарегистрирован для оплаты посредством удержания из зарплаты для размещения заказа
.Нет:	Если клиент не зарегистрирован для оплаты посредством удержания из зарплаты, система должна предложить клиенту следующие варианты: зарегистрироваться сейчас и продолжать размещать заказ, сделать заказ и самому получить его в кафетерии (без доставки) или выйти из системы
.Дата:	Система должна спрашивать клиента о дате заказа (см. BR-8)
.КрайнийСрок:	Если дата доставки заказа — текущий день, а крайний срок приема заказов уже прошел, то система должна известить клиента, что уже слишком поздно размещать заказ на сегодня. Клиент должен либо изменить дату, либо отменить заказ

Заказ.Доставка:	**Доставка или получение в кафетерии**
.Выбор:	Клиент должен указать, получит ли он заказ в кафетерии, или заказ должен быть доставлен
.Место:	Если заказ должен быть доставлен и все еще есть свободные интервалы времени доставки на дату заказа, клиент должен указать доступное место доставки
.Время:	Система должна известить клиента, если на дату заказа нет доступных интервалов времени доставки. Клиент должен либо отменить заказ, либо указать, что получит его в кафетерии
.Интервалы:	Система должна показывать свободные интервалы времени доставки на дату заказа Система должна позволять клиенту выбрать один из показанных интервалов доставки, сделать заказ без доставки или отменить заказ
Заказ.Меню:	**Просмотр меню**
.Дата:	Система должна отображать меню на выбранную дату
.Наличие:	Меню на выбранную дату должно показывать только те блюда, которые хотя бы в одном экземпляре есть в системе учета запасов кафетерия и могут быть доставлены
Заказ.Единицы:	**Заказ нескольких блюд и нескольких экземпляров одного блюда**
.Несколько:	Система должна позволять клиенту заказывать несколько одинаковых наборов блюд, вплоть до минимального числа любого из указанных блюд в меню, если таковое есть в заказе
.СлишкомМного:	Если клиент заказывает больше единиц одного блюда, чем в настоящее время указано в системе учета запасов кафетерия, система должна извещать клиента о максимальном количестве единиц того блюда, которое он может заказать
Заказ. Подтверждение:	**Подтверждение заказа**
.Отображение:	Когда клиент указывает, что не хочет больше заказывать никакие блюда, система должна отобразить заказанные блюда, цены на каждое из них и сумму к оплате, подсчитанную согласно BR-12
.Запрос:	Система должна предложить клиенту подтвердить заказ
.Ответ:	Если клиент может подтвердить, изменить либо отменить заказ
.Еще:	Система должна позволять клиенту заказывать дополнительные блюда на ту же или другие даты. Включение нескольких наборов блюд в один заказ регулируют бизнес-правила BR-3 и BR-4

Заказ.Оплата:	**Оплата заказа**
.Метод:	Когда клиент указывает, что закончил размещать заказы, система должна попросить пользователя выбрать метод оплаты
.Доставка:	См. бизнес-правило BR-11
.Самовынос:	Если клиент сам получит блюда в кафетерии, система должна предложить ему варианты оплаты: через удержание из зарплаты или наличными в кафетерии в момент получения заказа
.Удержание:	Если клиент подтвердил заказ и выбрал оплату через удержание из зарплаты, система должна выдать запрос на оплату системе расчета зарплат
.Да:	Если запрос на оплату принят, система должна вывести сообщение о подтверждении заказа с номером транзакции удержания из зарплаты
.Нет:	Если запрос на оплату не принят, система должна вывести сообщение с причиной отказа. Клиент должен либо отменить заказ, либо изменить метод оплаты на «наличные» и сделать запрос на получение заказа в кафетерии
Заказ.Завершение:	**После того как клиент подтвердил заказ, система должна сделать следующее как одну транзакцию**
.Сохранение:	Назначить заказу следующий доступный номер и сохранить заказ с начальным состоянием «Принят»
.Запасы:	Отправить сообщение инвентарной системе кафетерия, в котором указано количество единиц каждого блюда в заказе
.Меню:	Обновить меню на дату заказа, отражая возможные изменения в наличии блюд в системе управления запасами кафетерия
.Интервалы:	Обновить список остающихся доступными периодов доставки на дату заказа
.Клиент:	Отправить клиенту сообщение электронной почты с информацией о заказе и оплате
.Кафетерий:	Отправить сотрудникам кафетерия сообщение электронной почты с информацией о заказе
.Ошибка:	Если какой-либо шаг транзакции Заказ.Завершение не выполняется, система должна провести откат и сообщить пользователю, что заказ не был принят, с указанием причины неудачи

[***Примечание:*** *Функциональные требований по переупорядочению блюд и изменению и отмене блюд в этом примере не приводятся.*]

3.2. Заказ блюд из ресторанов

[*В этом примере детали не приводятся. Значительная часть функциональности, описанной в разделе «3.1. Заказ блюд из кафетерия» могут использоваться повторно, поэтому в этом разделе нужно указать только дополнительную функциональность, связанную с интерфейсом ресторанов.*]

3.3. Создание, просмотр, модификация и удаление подписки на стандартные блюда

[*В этом примере детали не приводятся.*]

3.4. Создание, просмотр, модификация и удаление меню кафетерия

[*В этом примере детали не приводятся.*]

4. Требования к данным

4.1. Логическая модель данных

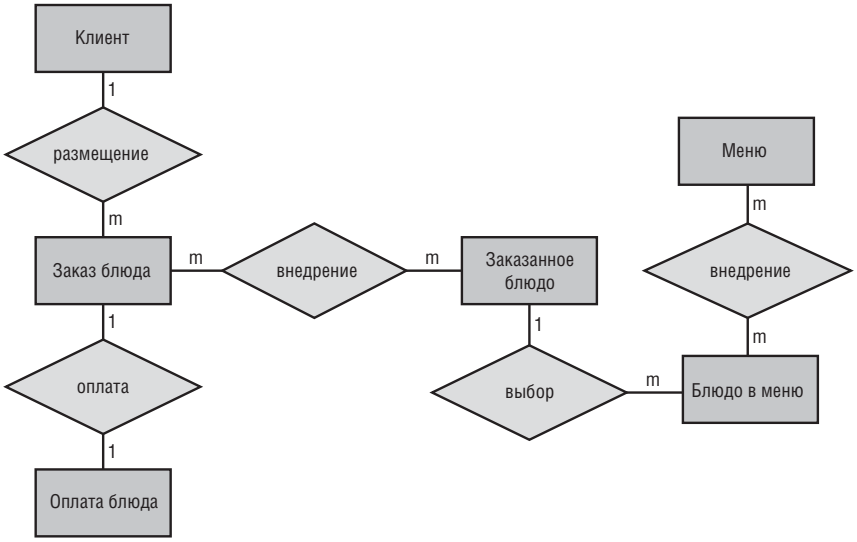


Рис. B-3. Фрагмент модели данных для выпуска 1.0 системы Cafeteria Ordering System

4.2. Словарь данных

Элемент данных	Описание	Структура или тип данных	Длина	Значения
указания по доставке	куда и кому должно быть доставлено блюдо, если его не забирают в кафетерии	имя клиента + телефон клиента + дата доставки блюда + пункт назначения поставки + временной интервал доставки		
пункт назначения поставки	строение или комната, куда нужно доставить заказанное блюдо	алфавитно-числовое значение	50	разрешены дефисы и запятые
временной интервал доставки	начало 15-минутного временного интервала на дату заказа блюда, во время которого должно быть доставлено блюдо	время	чч:мм	местное время; чч = 0-23 включительно; мм = 00, 15, 30 или 45
идентификатор сотрудника	корпоративный идентификатор сотрудника, который разместил заказ	целое	6	
описание блюда	описание блюда в меню	буквенное значение	100	
цена блюда	стоимость блюда до налога	числовое, рубли и копейки	рр.кк	
дата доставки блюда	дата, когда блюдо должно быть доставлено или получено в кафетерии	дата, дд.мм.гггг	10	по умолчанию — текущая дата, если она не выходит за рамки крайнего срока заказов, в противном случае — следующий день; не может быть раньше текущей даты

Элемент данных	Описание	Структура или тип данных	Длина	Значения
заказ блюда	подробности блюда, заказанного клиентом	номер заказа блюда + дата заказа + дата заказа блюда + 1: многие {заказанное блюдо} + указания по доставке + состояние заказа		
номер заказа блюда	уникальный идентификатор, назначаемый системой Cafeteria Ordering System каждому принятому заказу	целое	7	начальное значение — 1
состояние заказа	состоянием заказа, созданного клиентом	буквенное значение	16	незавершенный, принят, готовый, ожидающий доставки, доставлено, отменен
оплата блюда	информация о принятом системой Cafeteria Ordering System платеже за блюдо	размер платежа + способ оплаты + номер транзакции		
меню	список блюд, доступных для покупки на определенную дату	дата меню + 1: многие {блюдо меню}		
дата меню	дата на которую доступно определенное меню	дата, дд.мм.гггг	10	
блюдо меню	описание блюда в меню	описание блюда + цена блюда		
крайний срок заказов	время дня, до которого должны делаться заказы на эту дату	время, чч: мм	5	

Элемент данных	Описание	Структура или тип данных	Длина	Значения
дата заказа	дата на которую клиент размещает заказ	дата, дд.мм.гггг	10	
заказанное блюдо	одно из блюд, которое входит в заказ клиента	блюдо меню + заказанное количество		
клиент	сотрудник Process Impact, имеющий право заказывать еду	имя клиента + идентификатор клиента + телефон клиента + местоположение клиента + электронная почта клиента		
электронная почта клиента	корпоративный идентификатор клиента, который разместил заказ	алфавитно-числовое значение	50	
местоположение клиента	номера строения и комнаты сотрудника, который разместил заказ	алфавитно-числовое значение	50	разрешены дефисы и запятые
имя клиента	имя клиента, который разместил заказ	буквенное значение	30	
телефон клиента	телефон клиента, который разместил заказ	AAA-EEE-NNNN xXXXX где A — код территории, E — телефонная станция, N — номер и X — расширение	18	
размер платежа	общая цена заказа в рублях и копейках, вычисленная в соответствии с BR-12	числовое, рубль и копейки	рррр.кк	
способ оплаты	как клиент оплачивает заказ	буквенное значение	16	удержание из зарплаты, наличные, кредитная карта, дебетовая карта

Элемент данных	Описание	Структура или тип данных	Длина	Значения
заказанное количество	число единиц каждого блюда, который клиент включил в заказ	целое	4	по умолчанию — 1; максимум — количество, имеющееся в запасе
номер транзакции	уникальное последовательное число, которое Cafeteria Ordering System назначает каждой платежной операции	целое	12	

4.3. Отчеты

4.3.1. Отчет о заказанных блюдах

Идентификатор отчета	COS-RPT-1
Заголовок отчета	История заказов блюд
Цель отчета	Клиент хочет увидеть список всех блюд, которые он раньше заказывал в кафетерии или локальных ресторанах за определенный период времени вплоть до полугода до текущей даты, чтобы можно было повторно заказать понравившееся блюдо
Приоритет	Средний
Пользователи отчета	Постоянные клиенты кафетерия
Источники данных	База данных о ранее размещенных заказах блюд
Частота и использование	Отчет генерируется по запросу клиента. Данные в отчете статичны. Отчет отображается в окне веб-браузера пользователя на компьютере, планшете или смартфоне. Его можно распечатать, если устройство поддерживает печать
Время доступа	Готовый отчет должен отображаться в течение 3 секунд после его запроса
Визуальный макет	Альбомная ориентация
Верхний и нижний колонтитулы	Верхний колонтитул должен содержать заголовок отчета, имя клиента и заданный диапазон дат При печати в нижнем колонтитуле должен содержаться номер страницы

Идентификатор отчета	COS-RPT-1
Тело отчета	• Отображаемые поля и заголовки столбцов: • Номер заказа • Дата заказа блюда • Где заказано (кафетерий или название ресторана) • Заказанные блюда (список всех блюд в заказе с указанием их числа и цен) • Общая цена • Налог • Стоимость доставки • Итого (сумма стоимости блюд с налогом и доставки) • Критерий отбора: Диапазон дат, определенный клиентом, включая начальную и конечную даты • Критерий сортировки: Обратный хронологический порядок
Признак конца отчета	Нет
Интерактивность	Клиент может просматривать подробности ингредиентов и сведения о пищевой ценности для каждого блюда
Ограничения безопасности доступа	Клиент может просматривать историю только своих заказов

[*В этом примере не приводятся другие отчеты системы* Cafeteria Ordering System.]

4.4. Целостность, сохранение и утилизация данных

DI-1 Cafeteria Ordering System должна хранить заказы клиента на протяжении 6 месяцев в даты доставки.
DI-2 Cafeteria Ordering System должна хранить меню на протяжении года с даты, указанной на меню.

5. Требования к внешним интерфейсам

5.1. Пользовательские интерфейсы

UI-1 Экраны Cafeteria Ordering System должны соответствовать «*Process Impact Internet Application User Interface Standard, Version 2.0*» [4].
UI-2 Система должна обеспечивать ссылку на справку на каждой HTML-странице, объясняющую, как пользоваться этой страницей.
UI-3 Интернет-страницы должны предоставлять полную возможность навигации и выбор блюд только при помощи клавиатуры, в дополнение к использованию мыши и клавиатуры.

5.2. Интерфейсы ПО

SI-1 Система учета запасов кафетерия.
SI-1.1 Cafeteria Ordering System должна передавать количество единиц заказанных блюд системе учета запасов кафетерия через программный интерфейс.
SI-1.2 Cafeteria Ordering System должна опрашивать систему учета запасов кафетерия для определения наличия запрашиваемого блюда.
SI-1.3 Когда система учета запасов кафетерия сообщает Cafeteria Ordering System, что определенного блюда нет в наличии, Cafeteria Ordering System должна убирать это блюдо из меню на текущую дату.
SI-2 Система расчета зарплаты
Cafeteria Ordering System должна сообщаться с системой расчета зарплат через программный интерфейс, выполняя следующие операции.
SI-2.1 Позволять клиенту регистрироваться и отменять регистрацию для оплаты через удержания из зарплаты.
SI-2.2 Проверять, зарегистрирован ли клиент для оплаты посредством удержания из зарплаты.
SI-2.3 Проверять, может ли клиент регистрироваться для оплаты посредством удержания из зарплаты.
SI-2.4 Передавать запрос на оплату приобретенного набора блюд.
SI-2.5 Возвращать полностью или частично предыдущую оплату, если клиент отменил заказ, или не был удовлетворен им, или заказ не был доставлен согласно подтвержденным инструкциям по доставке.

5.3. Интерфейсы оборудования

Интерфейсы оборудования не выявлены.

5.4. Коммуникационные интерфейсы

CI-1 Cafeteria Ordering System должна отправлять клиенту сообщение электронной почты или СМС-сообщение (определяется параметрами учетной записи) с подтверждением принятия заказа, ценой и инструкциями по доставке.
CI-2 Cafeteria Ordering System должна отправлять клиенту сообщение электронной почты с или СМС-сообщение (определяется параметрами учетной записи) о любых проблемах, возникших с заказом или его доставкой после принятия заказа.

6. Атрибуты качества

6.1. Требования по удобству использования

USE-1 Система должна позволять клиенту извлечь ранее заказанное блюдо одной операцией.

USE-2 95% новых пользователей должны суметь успешно ввести заказ без ошибок с первой попытки.

6.2.Требования к производительности

PER-1 Система должна обслуживать всего 400 пользователей и 100 пользователей в период пиковой активности с 9:00 до 10:00 по местному времени, со средней продолжительностью сеанса 8 минут.
PER-2 Все веб-страницы, генерируемые системой, должны полностью загружаться не более чем за 4 секунды после запроса их по интернет-подключению со скоростью 20 Мбит/сек.
PER-3 Система должна выводить пользователю сообщение о подтверждении в среднем за 3 секунды и не более чем через 6 секунд после того, как пользователь отослал информацию системе.

6.3. Требования к безопасности

SEC-1 Все сетевые транзакции, включающие финансовую или поддающуюся учету личную информацию, должны быть зашифрованы согласно бизнес-правилу BR-33.
SEC-2 Пользователи обязательно регистрируются для входа в Cafeteria Ordering System для выполнения всех операций, кроме просмотра меню.
SEC-3 Система должна позволять только сотрудникам кафетерия, внесенным в список авторизованных менеджеров меню, создавать или изменять меню, согласно бизнес-правилу BR-24.
SEC-4 Система должна позволять клиентам просматривать только заказы, размещенные ими лично, но не другими клиентами.

6.4. Требования к защите

SAF-1 У пользователя должна быть возможность увидеть список всех ингредиентов каждого блюда, причем ингредиенты, известные тем, что могут вызывать аллергическую реакцию у 0,5% населения Северной Америки, должны быть выделены особо.

6.5. Требования к доступности

AVL-1 Cafeteria Ordering System должна быть доступна 98% времени между 5:00 и полуночью по местному времени и 90% времени между полуночью и 5:00 по местному времени, за исключением времени планового обслуживания.

6.6. Требование к надежности

ROB-1 Если соединение между пользователем и системой разрывается до того, как заказ подтвержден или отменен, Cafeteria Ordering System должна позволять пользователю восстановить незавершенный заказ и продолжить работу.

Приложение A. Модели анализа

На рис. B-4 показана диаграмма состояний, где отображено возможное состояние заказа блюд и его возможные изменения.

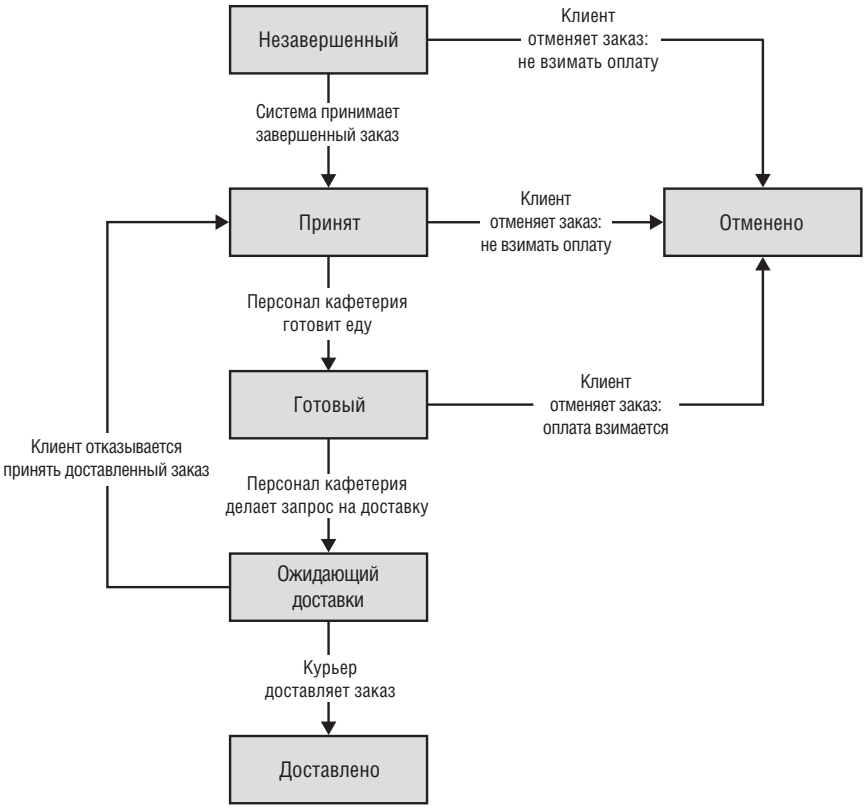


Рис. B-4. Диаграмма состояний для состояния заказов блюд

Бизнес-правила

[***Примечание:*** *Нижеследующее — пример отдельного перечня бизнес-правил.*]

Идентификатор	Определение правила	Тип правила	Статичное или динамическое	Источник
BR-1	Периоды доставки — это 15-минутные интервалы, начинающиеся каждую четверть часа	Факт	Динамическое	Менеджер кафетерия
BR-2	Доставка всех заказов должна быть завершена между 10:00 и 14:00 по местному времени	Ограничение	Динамическое	Менеджер кафетерия

Идентификатор	Определение правила	Тип правила	Статичное или динамическое	Источник
BR-3	Все блюда из одного заказа должны доставляться в одно место	Ограничение	Статическое	Менеджер кафетерия
BR-4	Все блюда из одного заказа должны быть оплачены одним и тем же методом	Ограничение	Статическое	Менеджер кафетерия
BR-8	Блюда должны быть заказаны не более, чем за 14 календарных дней до даты доставки	Ограничение	Динамическое	Менеджер кафетерия
BR-11	Если заказ должен доставляться, клиент должен оплатить его посредством удержания из зарплаты	Ограничение	Динамическое	Менеджер кафетерия
BR-12	Стоимость заказа подсчитывается как сумма цен единиц каждого блюда, умноженных на количество заказанных единиц этого блюда, плюс соответствующий налог с продаж, плюс плата за доставку, если заказ доставляется в пункт, расположенный вне зоны бесплатной доставки	Вычисление	Динамическое	Политика кафетерия, налоговые законы штата
BR-24	Только работники кафетерия, назначенные менеджером кафетерия менеджерами меню, могут создавать, изменять или удалять меню кафетерия	Ограничение	Статическое	Политика кафетерия
BR-33	Передача данных по сети, включающая финансовую или поддающуюся учету личную информацию, должна проходить с использованием 256-разрядного шифрования	Ограничение	Статическое	Политика безопасности компании

Идентификатор	Определение правила	Тип правила	Статичное или динамическое	Источник
BR-86	Только штатные сотрудники могут регистрироваться для совершения каких-либо покупок в компании посредством удержания из зарплаты	Ограничение	Статическое	Финансовый директор компании
BR-88	Сотрудник может зарегистрироваться для оплаты питания в кафетерии посредством удержания из зарплаты, если не более 40% его начисленной зарплаты удерживается в настоящее время по другим причинам	Ограничение	Динамическое	Финансовый директор компании

Словарь терминов

CRUD-матрица ~ CRUD matrix — таблица, связывающая действия системы с элементами данных, чтобы показать, где каждый элемент создается (Created), читается (Read), обновляется (Updated) и удаляется (Deleted).

Planguage — основанный на ключевых словах язык, предложенный Томом Гилбом (Tom Gilb) и позволяющий создать точную и количественно оцениваемую спецификацию требований.

Swimlane-диаграмма ~ diagram — модель анализа, показывающая последовательные шаги потока бизнес-процессов предлагаемой программной системы. Процесс разбивается на визуальные компоненты, называемые дорожками, которые показывают системы или действующие лица, выполняющие эти шаги.

TBD — сокращение от To Be Determined. Служит для отметки неясностей или пропусков, которые надо заполнить, в информации требований.

UML (Unified Modeling Language) — набор стандартной нотации для создания различных визуальных моделей систем, особенно в объектно-ориентированном программировании.

Альтернативное направление ~ alternative flow — направление, учитывающее вариант использования, которое ведет к успеху (достижение цели действующего объекта), но которое подразумевает отклонение от нормального направления в специфике задач или при взаимодействии объекта с системой.

Анализ первопричин ~ root cause analysis — действия, которые предпринимаются для поиска основных причин, вызывающих наблюдаемые проблемы.

Анализ расхождение ~ gap analysis — сравнение текущего состояния и альтернативного или возможного состояния системы, процесса или другого аспекта бизнес-ситуации для выявления значительных расхождений между ними.

Анализ требований ~ requirements analysis — процесс классификации информации, касающейся требований, по различным категориям, оценка требований для определения желаемого качества, представление требований в различных формах, выделение детальных требований из требований более высокого уровня, обсуждение приоритетов требований и т. д.

Архитектура ~ architecture — структура системы, включающая все ПО, оборудование и людей, из которых она состоит, интерфейсы и взаимосвязи между этими компонентами и поведение компонентов, видимое другим компонентам.

Атрибут качества ~ quality attribute — вид нефункционального требования, описывающего характеристику сервиса или производительности продукта. Примеры

атрибутов качества: удобство и простота использования, легкость перемещения, легкость в эксплуатации, целостность, надежность, эффективность и устойчивость к сбоям. В требованиях описаны рамки атрибутов качества, до которых продукт демонстрирует желаемые характеристики.

Атрибут требования ~ requirement attribute — описательная информация о требованиях, которая дополняет описание желаемой функциональности продукта. К ней относятся источники данных, логические обоснования, приоритеты, ответственные лица, номера версий и выпусков.

Бизнес-аналитик ~ business analyst — роль члена команды по разработке требований, основная обязанность которой — работа с заинтересованными лицами над выявлением, анализом, определением, утверждением и управлением требованиями в проекте. Эта роль также может называться *аналитик требований*, *системный аналитик*, *разработчик требований* и просто *аналитик*.

Бизнес-правило ~ business rule — политика, предписание, стандарт, правило или вычислительная формула, определяющая или ограничивающая некоторые стороны бизнес-процессов.

Бизнес-требования ~ business requirement — объем информации, который в совокупности описывает потребность, которая инициирует один или больше проектов, призванных предоставить решение и получить требуемый конечный бизнес-результат. Бизнес-требования включают бизнес-возможности, бизнес-цели, метрики успеха, концепция и границы и ограничения.

Бизнес-цель ~ business objective — финансовая или нефинансовая выгода, которую организация ожидает получить в результате реализации проекта или другой инициативы.

Блок-схема ~ flowchart — модель, которая показывает этапы процесса и точки принятия решений в логике процесса или программы. Аналогична диаграмме взаимодействия.

Большие данные ~ big data — обычно описывают массив данных, отличительные особенности которых — большой объем (много данных), высокая скорость (данные быстро поступают в организацию) и/или высокая сложность (данные очень разнородны). Управление большими данными предусматривает обнаружение, сбор, хранение и обработку больших объемов данных быстро и эффективно.

Бумажный прототип ~ paper prototype — непрограммная модель пользовательского интерфейса ПО с недорогими, несложными в исполнении эскизами экрана.

Вариант использования ~ use case — описание набора логически связанных возможных взаимодействий действующего лица и системы, которые дают результат, ценный для действующего лица. Может включать много сценариев.

Взаимосвязь «расширить» ~ extend relationship — конструкция, в которой альтернативное направление ответвляется от нормального направления в отдельный «расширенный» вариант использования.

Владелец продукта ~ product owner — роль, обычно в команде проекта гибкой разработки, представляющая клиента и отвечающая за определение концепции продукта, предоставление границ и ограничений проекта, определение приоритетов запаса продукта и принятие решений по продукту.

Внешняя сущность ~ external entity — объект в контекстной диаграмме или диаграмма потока данных, представляющая класс пользователей, действующее лицо, программную или аппаратную систему и являющийся внешним к описываемой системе, но так или иначе взаимодействует с ней. Также называется оконечным устройством.

Водопадный жизненный цикл проекта ~ waterfall development life cycle — модель процесса разработки ПО, в которой различные действия с требованиями, дизайном, кодированием, тестированием и развертыванием выполняются последовательно, с небольшим наложением или итерациями.

Встроенная система ~ embedded system — система, содержащая аппаратные компоненты, управляемые ПО, работающем на выделенном компьютере, являющемся частью более крупного продукта.

Выходное условие ~ postcondition — условие, описывающее состояние системы после успешного завершения варианта использования.

Выявление требований ~ requirements elicitation — процесс определения требований из различных источников посредством интервью, семинаров, анализа задач, рабочих потоков и документов и других механизмов.

Гибкая разработка ~ agile development — методы разработки ПО, характеризующиеся постоянным взаимодействием между разработчиками и клиентами. К методам гибкой разработки относятся экстремальное программирование (Extreme Programming), Scrum, разработка, управляемая функциональностью (Feature-Driven Development), бережливая разработка программного обеспечения (Lean Software Development) и Kanban.

Граница проекта ~ scope — часть концепции конечного продукта, реализуемой в ходе текущего проекта. Определяет границу между тем, что входит и что не входит в проект, в котором создается определенный выпуск или итерация.

Действующее лицо ~ actor — лицо, играющее конкретную роль, программная система или аппаратное устройство, которое взаимодействует с системой для достижения полезных целей. Также называется *ролью пользователя*.

Дерево решений ~ decision tree — модель анализа, которая графически показывает действия системы в ответ на все комбинации набора факторов, которые влияют на поведение части системы.

Дерево функций ~ feature tree — модель анализа, отображающая функции, запланированные для продукта, в виде иерархического дерева и отображающая до двух уровней подчиненных функций в каждой функции.

Дефект требования ~ requirement issue — проблема, открытый вопрос или решение относительно требования. Это могут быть элементы, помеченные как «TBD» (to be determined — необходимо определить), отложенные решения, необходимая информация, неразрешенные конфликты и т.п.

Диаграмма (или машина) состояний ~ state machine diagram — модель анализа, показывающая представления различных состояний, которые могут принимать объекты системы на протяжении своего жизненного цикла в ответ на то или иное событие или отображающая возможные состояния системы в целом. Похожа на диаграмму перехода состояний.

Диаграмма «сущность–связь» ~ entity-relationship diagram — модель анализа, которая показывает логические взаимосвязи между парами объектов. Используется для моделирования данных.

Диаграмма вариантов использования ~ use case diagram — модель анализа с указанием действующих лиц, которые могут взаимодействовать с системой для выполнения задач, и различные варианты использования, в которых может участвовать действующее лицо.

Диаграмма взаимодействия ~ activity diagram — аналитическая модель, которая позволяет динамически представить систему, посредством изображения потока процессов от одной функции к другой. Схожа с блок-схемой.

Диаграмма классов ~ class diagram — аналитическая модель, которая показывает набор классов системы или определенной предметной области, их интерфейсы и взаимосвязи.

Диаграмма перехода состояний ~ state-transition diagram — модель анализа, показывающая возможные состояния системы или объектов в ней, разрешенные переходы между ними и условия и/или события, инициирующие каждый переход. Аналогична диаграмме или машине состояний.

Диаграмма потока данных ~ data flow diagram — модель анализа, описывающая процесс, хранилища данных, внешние сущности и потоки, характеризующие поведение данных, проходящих через бизнес-процессы или программные системы.

Документ о концепции и границах ~ vision and scope document — документ, в котором определены бизнес-требования к новой системе, в том числе положения о концепции продукта и описания границы проекта.

Документы процесса ~ process assets — документы, такие как шаблоны, формы, списки, политики, процедуры, описание процессов и примеры рабочих продуктов, которые собраны для эффективного применения в организации с целью улучшить приемы разработки ПО.

Допущение ~ assumption — положение, которое считается верным в отсутствие доказательств или точных знаний.

Зависимость ~ dependency — зависимость проекта от внешних факторов, событий или групп, находящихся вне зоны контроля.

Заинтересованное лицо ~ stakeholder — это человек, группа или организация, которая активно задействована в проекте, подвержена влиянию процесса или результата или может влиять на процесс или результат.

Исключение ~ exception — условие, которое может помешать успешному завершению варианта использования. Если некоторые возвратные механизмы не работают, то выходные условия варианта использования не достигаются и желаемая цель не достигается.

Итерация ~ iteration — непрерывный период разработки, обычно от одного до четырех недель, во время которого команда разработки реализует определенный набор функциональности, выбранной из резерва продукта, или базовой версии требований к продукту.

Каркас ~ wireframe — разновидность одноразового прототипа, который используется для предварительного дизайна веб-страниц.

Карта диалоговых окон ~ dialog map — модель анализа, описывающая архитектуру пользовательского интерфейса, показывая видимые диалоговые элементы и навигацию между ними.

Карта экосистемы ~ ecosystem map — аналитическая модель, которая показывает набор классов системы или определенной предметной области, их интерфейсы и взаимосвязи. В отличие от контекстных диаграмм, карта экосистемы показывает системы, имеющие отношения друг с другом, даже если между ними нет интерфейса.

Класс пользователей ~ user class — группа пользователей системы, имеющих схожие требования к системе. Члены пользовательского класса функционируют как действующие лица при взаимодействии с системой.

Класс ~ class — описание набора объектов, имеющих общие свойства и поведение, которые стандартным образом соотносятся с элементами реального мира (людьми, местами или вещами) в бизнесе или определенной предметной области.

Клиент ~ customer — человек или организация, получающая от продукта прямую или косвенную выгоду. Клиенты это заинтересованные в проекте лица, запрашивающие, оплачивающие, выбирающие, определяющие, использующие и получающие результаты работы программного продукта.

Количество элементов ~ cardinality — количество элементов данных объектов или данных, которые связаны с элементами других объектов или данных. Например, «один к одному», «один ко многим» или «многие ко многим».

Контекстная диаграмма ~ context diagram — аналитическая модель, которая описывает абстрактную систему высокого уровня. Контекстная диаграмма определяет внешние для системы объекты, которые взаимодействуют с ней, но не отображает ничего из внутренней структуры или поведения системы.

Концепция ~ vision — утверждение, описывающее стратегический принцип конечной цели и формы новой системы.

Критерий приемлемости ~ acceptance criteria — условия, которым продукт должен удовлетворять, чтобы его приняли пользователи, клиенты или другие заинтересованные лица.

Матрица отслеживания связей требований ~ requirements traceability matrix — таблица, отображающая логические связи между функциональными требованиями и другими системными артефактами, в том числе функциональными требованиями, пользовательскими требованиями, бизнес-требованиями, элементами архитектуры и дизайна, модулями кода, тестами и бизнес-правилами.

Модель ~ mock-up — частичная или возможная реализация пользовательского интерфейса для систем ПО. Применяется для оценки легкости и простоты использования, а также завершенности и корректности требований. Может представлять собой программу или прототип на бумаге. Также называется *горизонтальным прототипом*.

Модель бизнес-целей ~ business objectives model — визуальное представление иерархии бизнес–задач и бизнес–целей.

Нефункциональное требование ~ nonfunctional requirement — описание присущих свойств или характеристик, которые система ПО должна демонстрировать, или ограничения, которые она должна соблюдать.

Нормальное направление ~ normal course — последовательность действий, заданная по умолчанию в варианте использования, которая ведет к удовлетворению выходных условий этого варианта использования или достижению целей пользователей. Другие названия: нормальное направления развития, базовый поток, нормальная последовательность, основной успешный сценарий и счастливый путь (happy path).

Ограничение ~ constraint — налагается на доступные разработчику возможности дизайна или конструирования продукта. Другие типы ограничений могут ограничить возможности, доступные для менеджеров проектов. Бизнес-правила часто накладывают ограничения на бизнес-операции, а значит, на программные системы.

Одноразовый прототип ~ throwaway prototype — прототип, который создается с расчетом, что после его использования для уточнения и утверждения требований и вариантов дизайна он будет выброшен.

Оперативный профиль ~ operational profile — комплект сценариев, который представляет ожидаемые случаи применения ПО.

Организатор мероприятия ~ facilitator — лицо, ответственное за планирование и работу группы, например работу семинара по выявлению требований.

Основная версия требований ~ requirements baseline — зафиксированный в определенный момент времени, согласованный, просмотренный и одобренный набор требований, обычно определяющих определенный выпуск продукта или итерацию разработки. Служит основой для дальнейшей разработки.

Отношение включения ~ include relationship — конструкция, в которой несколько шагов, повторяющихся во многих вариантах использования, выделяются в отдельный вложенный вариант использования, который вызывается по мере необходимости.

Отслеживание ~ tracing — процесс определения логических связей между одним элементом системы (вариантом использования, функциональными требованиями, бизнес-правилами, компонентами дизайна, фрагментами кода, тестами и т. д.) и другим.

Панель мониторинга ~ dashboard — изображение на экране или в печатном документе с множественными текстовыми и/или графическими представлениями данных, предоставляющее консолидированное многомерное представление происходящего в организации или процессе.

Пилотная версия ~ pilot — контролируемое выполнение нового решения (такого как процесс, инструмент, программная система или учебный курс) для оценки его работы в реальных условиях и готовности к развертыванию.

Повторное использование требований ~ requirements reuse — использование существующих требований во многих системах, отличающихся одинаковой функциональностью.

Пользователь ~ user — клиент, который взаимодействует с системой непосредственно или косвенно (например, пользуется результатами работы системы, хотя не генерирует эти результаты). Также называется конечным пользователем.

Пользовательская история ~ user story — способ выражения пользовательских требований в проектах гибкой разработки в форме одного или двух предложений,

формулирующих потребность пользователя или описывающих единицу необходимой функциональности, а также говорящих о пользе, какую эта функциональность приносит пользователю.

Пользовательское требование ~ user requirement — цель и задача, которую пользователи должны иметь возможность выполнять с системой, или положения об ожиданиях пользователей о качестве системы. Пользовательские требования обычно представляются в виде вариантов использования, пользовательских историй и сценариев.

Поток процесса ~ process flow — последовательные шаги бизнес-процесса или операций предложенной программной системы. Обычно предоставляется с применением диаграммы взаимодействия, блок-схемы, swimlane-диаграммы или другой нотации моделирования.

Правило решения ~ decision rule — согласованный способ выработки единого решения в группе людей.

Предварительные условия ~ precondition — условия, которые должны быть удовлетворены, или состояние, в котором система должна пребывать, чтобы мог начаться вариант использования.

Приемочный тест ~ acceptance test — тест для проверки ожидаемых вариантов использования с целью определения приемлемости ПО. Используется в проектах гибкой разработки для представления подробностей пользовательских историй и определения правильности их реализации.

Приоритизация, определение приоритетов, расстановка приоритетов ~ prioritization — акт определения того, какие требования программного продукта наиболее важны для достижения бизнес-успеха и в каком порядке должны реализовываться требования.

Проверка ~ verification — процесс оценки рабочего продукта, позволяющий определить, удовлетворяет ли он спецификации, на основе которой создан. Обычно формулируется в виде вопроса: «Правильно ли мы создаем продукт?»

Продукт ~ product — конечный результат разработки, выполняемой в рамках проекта. В этой книге используются также термины-синонимы «приложение», «система» и «решение».

Проект с чистого листа ~ green-field project — проект, в котором разрабатывается новое ПО или новая система.

Прототип ~ prototype — частичная, предварительная или возможная реализация программы. Применяется для исследования и утверждения требований, а также для разработки приемов. Прототипы бывают эволюционные, одноразовые, бумажные, горизонтальные и вертикальные.

Процедура ~ procedure — пошаговое описание направления действия для выполнения и завершения конкретной работы.

Процесс ~ process — последовательность действий, выполняемых для достижения конкретной цели. Описание процесса представляет собой документированное определение этих действий.

Процесс создания требований, процесс построения требований ~ requirements engineering — область, которая охватывает все стороны жизненного цикла проек-

та, связанные с необходимыми возможностями и атрибутами продукта. Состоит из разработки требований и управления требованиями. Считается подобластью процессов построения системы и ПО.

Рабочий продукт ~ work product — любой промежуточный или окончательный результат, созданный в проекте разработки ПО.

Разработка требований ~ requirements development — процесс определения границ проекта, классов и представителей пользователей, выявления, анализа, спецификации и утверждения требований. Результатом этого процесса считается основная версия требований, в которой указано, что за продукт должен быть построен.

Расползание границ проекта ~ scope creep — условия, при которых границы проекта неконтролируемо расширяются на протяжении всего процесса.

Распределение требований, назначение требований ~ requirements allocation — процесс распределения системных требований по различным архитектурным и компонентным подсистемам.

Резерв продукта ~ product backlog — в проекте гибкой разработки, распределенные по приоритетам список еще не реализованных задач проекта. Резерв может содержать пользовательские истории, бизнес-процессы, запросы на изменение и разработку инфраструктуры. Рабочие элементы из резерва назначаются на будущие итерации на основе их приоритетов.

Ретроспектива ~ retrospective — рецензирование, в котором участники анализируют действия и результаты проекта с целью определения путей повышения успешности последующих проектов.

Рецензирование ~ peer review — действия, предпринимаемые одной или несколькими лицами (не авторами продукта), для исследования продукта с целью обнаружить возможные дефекты и улучшить возможности.

Решение ~ solution — все компоненты, которые должны быть созданы в процессе реализации проекта для достижения бизнес-целей, определенных организацией, в том числе ПО, оборудование, бизнес-процессы, руководство пользователя и обучение.

Риск ~ risk — условие, которое может привести к потере или иным образом поставить под угрозу успех проекта.

Серийные продукты, коммерческие готовые продукты ~ COTS (commercial off-the-shelf) product — готовый пакет ПО, приобретаемый у поставщика. Применяется как готовое решение или интегрируется, настраивается и расширяется в соответствии с потребностями клиента для удовлетворения нужд последнего.

Система ~ system — продукт, содержащий много программных или аппаратных подсистем. В общеупотребимом смысле используется в этой книге в отношении приложения, продукта и решения для обозначения любого содержащего ПО результата, создаваемого командой.

Система бизнес-аналитики ~ business analytics system — программная система, служащая для преобразования больших и сложных наборов данных в осмысленную информацию, на основе которой можно принимать решения.

Система реального времени ~ real-time system — аппаратная или программная система, которая должна реагировать в четко определенное время на заданные события.

Системное требование ~ system requirement — требование верхнего уровня к продукту, состоящему из многих подсистем, которые могут представлять собой ПО или совокупность ПО и оборудования

Словарь данных ~ data dictionary — набор определений элементов данных, структуры и атрибутов, относящихся к определенной предметной области.

Событие ~ event — инициирующее или стимулирующее событие, которое происходит в системной среде, например поведение функции или изменение состояния.

Совет по управлению изменениями ~ change control board — группа сотрудников, отвечающая за решение о принятии или отклонении предлагаемых изменений в требованиях к ПО.

Спецификация требований к продукту ~ software requirements specification — набор функциональных и нефункциональных требований к продукту ПО.

Сторонник продукта ~ product champion — назначенный представитель отдельного класса пользователей, который предоставляет пользовательские требования представляемых им групп пользователей.

Сущность ~ entity — элемент области бизнеса, данные о котором собираются и сохраняются.

Схема требования ~ requirement pattern — систематический подход к определению определенного типа требований.

Сценарий ~ scenario — описание взаимодействия пользователя и системы с целью достижения некоторой цели. Пример работы с системой. Один из путей развития варианта использования.

Таблица «событие–реакция» ~ event-response table — перечень внешних или зависящих от времени событий, которые могут влиять на систему, и описание того, как система будет отвечать на каждое из них.

Таблица решения ~ decision table — модель анализа в виде матрицы, где показаны все комбинации значений для наборов факторов, которые влияют на поведение части системы, и определены ожидаемые действия системы в ответ на каждую комбинацию.

Таблица состояний ~ state table — модель анализа, показывающая в виде матрицы состояния, в которых может находиться система или ее объекты, а также какие возможны переходы между состояниями.

Требование ~ requirement — документ, где указаны потребности или цели пользователей либо условия и возможности, которыми должен обладать продукт, чтобы удовлетворить такие возможности или цели. Свойство, которым должен обладать продукт, чтобы представлять ценность для заинтересованного лица.

Требования для интерфейса внешнего устройства ~ external interface requirement — описание интерфейса между системой ПО и пользователем, другой системой ПО или оборудованием.

Требования-«бантики», украшательство ~ gold plating — не являющаяся необходимой или избыточно сложная функциональность, запланированная и разработанная для продукта, иногда без одобрения клиента.

Управление требованиями ~ requirements management — работа с определенным набором требований к продукту, начиная от процесса разработки продукта и заканчивая поддержкой действующего продукта. Управление подразумевает отслеживание состояния продукта, управление изменениями требований и версиями спецификаций требований, а также отслеживание требований до других требований и элементов системы.

Утверждение ~ validation — процесс оценки рабочего продукта, позволяющий определить, действительно ли он удовлетворяет потребности клиента. Обычно формулируется в виде вопроса: «Создаем ли мы правильный продукт?»

Функциональная точка ~ function point — мера размера ПО, основанная на числе и сложности внутренних логических файлов, файлов интерфейса внешнего устройства, вводимых извне данных, результатов и запросов.

Функциональное требование ~ functional requirement — описание поведения системы в определенных условиях.

Функция ~ feature — одна или несколько логически связанных возможностей системы, которые представляют ценность для пользователя и описаны рядом функциональных требований

Цикл разработки ПО ~ software development life cycle — последовательность действий, в которой ПО определяется, конструируется, строится и проверяется.

Шаблон ~ template — образец, который используется в качестве руководства при создании всеобъемлющей документации или других элементов.

Эволюционный прототип ~ evolutionary prototype — полностью функциональный прототип, построенный как скелет или некая стадия конечного продукта, которые постепенно будут «обрастать мясом» по мере прояснения требований.

Экспериментальный образец ~ proof of concept — прототип, реализующий часть содержащей программную часть системы и охватывающий много уровней архитектуры. Применяется для оценки технической осуществимости и производительности. То же самое, что *вертикальный прототип*.

Эксперт предметной области ~ subject matter expert — лицо, имеющее обширный опыт и знания в предметной области и считающееся полномочным источником информации о предметной области.

Экспертиза ~ inspection — тип рецензирования, когда члены специально созданной команды в определенном и строгом порядке исследуют рабочий продукт на предмет выявления дефектов.

Эпика ~ epic — пользовательская история в проекте гибкой разработки, которая слишком большая, чтобы ее можно было реализовать в одной итерации разработки. Эпика разбивается на более мелкие истории, каждая из которых может быть реализована в одной итерации.

Эпилог

Для успешной разработки требований необходимо:

- вовлекать представителей клиентов как можно раньше и шире;
- разрабатывать требования итеративно и поступательно;
- представлять требования различными способами, чтобы удостовериться, что все понимают их;
- убедиться, что все группы заинтересованных лиц считают требования полными и корректными;
- найти подходящие вспомогательные технологии и приемы, которые позволят получить единое представление для всех заинтересованных лиц и обеспечить целостность требований;
- контролировать внесение изменений в требования.

Успех процесса улучшения разработки ПО зависит от многих причин:

- обращайте внимание на несколько проблем;
- установите ясные цели и разработайте план действий для улучшения процесса;
- выявите связанные с людьми и корпоративной культурой факторы, связанные с изменениями в организации;
- убедите всех, что процесс улучшений следует рассматривать как стратегическую инвестицию в успех бизнеса.

Помните об этих принципах, когда будете строить «дорожную карту» реорганизации процесса разработки требований. Выбирайте такие приемы, которые подходят для вашей организации и соответствуют духу вашей команды. Если вы активно применяете рекомендованные приемы и полагаетесь на здравый смысл, вам удастся значительно улучшить процесс обработки требований и получить все преимущества и выгоды, которые за этим следуют. И запомните, что без прекрасных требований разрабатываемый продукт выглядит, как коробка с шоколадом: вы не знаете, что обнаружите, открыв ее.

Об авторах

Карл Вигерс (Karl Wiegers)

Карл И. Вигерс — главный консультант компании Process Impact, которая занимается консультированием и подготовкой специалистов в области разработки ПО и находится в городе Портленд в Орегоне. Его область интересов включает разработку требований, рецензирование, управление проектами и совершенствование процессов. Ранее почти 18 лет Карл работал в Eastman Kodak Company, где занимался исследованием фотографического процесса, разработкой и управлением ПО, а также руководил процессом улучшения качества продукта. Карл получил степень доктора органической химии в Университете Иллинойса.

Карл — автор многих книг и статей по разработке ПО, химии, работе над собой и военной истории. Из-под его пера вышли два предыдущих издания книги «Разработка требований к программному обеспечению», а также «More About Software Requirements» (Microsoft Press, 2006), «Practical Project Initiation» (Microsoft Press, 2007), «Peer Reviews in Software» (Addison-Wesley, 2001) и «Creating a Software Engineering Culture» (Dorset House Publishing, 1996). Он провел более 300 семинаров и курсов по разработке требований. Связаться с Карлом можно через сайт *www.processimpact.com* или *www.karlwiegers.com*. (Автор фото: Эмили Даун (Emily Down) из компании Jama Software).

Джой Битти (Joy Beatty)

Джой Битти — вице-президент в Seilevel, компании (г. Остин, Техас), специализирующейся в области профессиональных услуг и помогающей своим клиентам реструктурировать свои процессы разработки ПО. Обладая 15-летним опытом бизнес-анализа, Джой развивает новые методы и помогает клиентам реализовывать лучшие приемы, позволяющие совершенствовать выявления и моделирование требований. Она помогает компаниям из списка Fortune 500 создавать образцовые центры бизнес-анализа. Джой читала курсы для тысяч бизнес-аналитиков и носит звание Certified Business Analysis Professional (CBAP). Джой получила звание бакалавра компьютерных наук и математики в университете г. Пардью.