Стив Макконнелл

Еще более эффективный Agile

Питер
2021

Отзывы о книге «Еще более эффективный Agile»

Если вы менеджер, только внедряете методологию Agile или ищете способы повысить эффективность используемых Agile-методов, то здесь найдете полезные советы, основанные на тщательном исследовании и богатом опыте.
Шахида Низар, ведущий специалист по инжинирингу, Google

Благодаря этой книге самые свежие концепции из мира гибкой разработки становятся доступными для ведущих специалистов, в то же время книга позволяет посмотреть на проблему с точки зрения бизнеса и ориентации на ценности.
Джон Рейндерс, вице-президент по стратегии научно-исследовательских работ, управлению программами и наукам о данных, Alexion Pharmaceuticals

Эта книга дает ясное представление о том, что Agile — это набор методов, который привносит ценность вашему бизнесу, а не только лишь набор предписанных ритуалов.
Гленн Гудрич, вице-президент по разработке продуктов, Skookum

28 ключевых принципов этой книги представляют собой превосходную шпаргалку, пожалуй, наиболее ценных уроков, усвоенных в отрасли разработки программных продуктов за последние четыре десятилетия. Книга сосредоточена на этих принципах таким образом, что переплетает теорию и практику.
Ксандер Бота, технический директор, Demonware

Книга дает ясное представление о том, что методология Agile (при правильном подходе) может быть неожиданно эффективной в случаях, которые на протяжении всего времени предполагали модель последовательной разработки, например когда предсказуемость критически важна или важно следование нормативам.
Чарльз Дэвис, главный технический директор, TomTom

Книга легко читается как технарями, так и управленцами, чем преодолевается разрыв в общем понимании Agile.
Сунил Крипалани, директор по цифровым технологиям, OptumRx

В этой книге даже эксперты по Agile смогут найти пищу для размышлений, которая утвердит их желание применять данную методологию.
Стефан Ландвогт, ведущий инженер-программист, Microsoft

Многие подходы, применяемые в Agile, идеалистичны и не оправдывают себя в реальности. Эта книга — прекрасный ориентир в лабиринте,

блуждания в котором не избежать при внедрении Agile. Из нее можно узнать, что нужно искать (проверять) и что с этим делать, когда нашли (адаптироваться).
Ильхан Дилбер, директор по тестированию и контролю качества, CareFirst

На удивление в этой книге приводятся не догматические положения Agile. Здесь объясняется, как применять методы в соответствии с потребностями вашего бизнеса.
Брайан Дональдсон, президент Quadrus

Предсказуемость часто (ошибочно) рассматривают как то, чего нужно добиваться от Agile, а не то, что присуще Agile самому по себе. Методики, приведенные в этой книге, — отличные рекомендации для того, чтобы развеять этот миф.
Лиза Форсайт, старший директор, Smashing Ideas

Эта книга, будучи емкой и практической, сосредоточена на том, чтобы оправдать обещание, отраженное в ее названии. Она особенно ценна для ведущих разработчиков, которые хотят повысить эффективность от внедрения Agile. Также будет полезна для руководителей, которые только начинают или подумывают о переходе на Agile.
Дэвид Уайт, консультант, Calaveras Group

В книге дается целостное представление о том, как эффективно внедрить Agile и со временем улучшить показатели уже после внедрения. Во многих книгах рассказано, как начать, но лишь в немногих из них говорится о том, как продолжить работу после внедрения и какие инструменты использовать.
Эрик Апчерч, ведущий разработчик-архитектор, Synaptech

Книга объединяет все особенности создания современных, преимущественно программных, систем: технических, управленческих, организационных, культурных и человеческих — в понятное, последовательное и осуществимое целое, основанное на реальном опыте.
Джованни Аспрони, главный консультант, Zublke Engineering Ltd

Потрясающие советы о том, как справляться с крупными организационными проблемами так, чтобы получить пользу от внедрения Agile. Например, границы Agile, смена моделей управления, управление портфолио, а также предсказуемость и управление.
Хиранья Самарасекера, вице-президент по инжинирингу, Sysco LABS

Выразительная и впечатляющая презентация, которая предлагает нечто ценное каждому сотруднику и каждой компании, в первую очередь там, где программное обеспечение является ключевой составляющей

выполняемой работы, но многие концепции в целом можно применить в любом бизнесе.
Барбара Тэлли, директор, аналитик бизнес-систем, Epsilon

Авторитетный источник сведений, лучших методов, челленджей, инструкций и других источников знаний. Это настольная книга для меня и моей команды. Мне иногда сложно объяснить Agile-методы и способы повышения их эффективности. В этой книге даны блестящие пояснения.
Грэм Хэйторнтвейт, вице-президент по технологиям, Impero Software

Книга учит нас смотреть на Agile как на набор инструментов, которые можно использовать выборочно по ситуации, а не как «пан или пропал».
Тимо Киссел, старший вице-президент по инжинирингу, Circle Media

Замечательная книга, которая наконец отвечает на вопрос «Зачем применять Agile?».
Дон Шафер, главный специалист по безопасности, охране труда и окружающей среды, Athens Group

Тем, кто только начинает знакомиться с Agile, советую сразу перейти к главе «Еще более эффективное внедрение». Мне доводилось видеть много организаций, мчавшихся к Agile на всех парах, но не заложивших должного фундамента, чтобы преуспеть в этом.
Кевин Тейлор, старший, архитектор систем облачных вычислений, Amazon

Книга увидела свет, чтобы мы могли узнать, что работает и что другие считают полезным, в том числе в щепетильных вопросах, касающихся культуры, людей и команд, а также процесса и архитектуры. Несмотря на размер, книга охватывает на удивление многое!
Майк Блэксток, технический директор, Sense Tecnic Systems

Честный взгляд на 20-летнюю методологию и, наверное, первая книга, которая напрямую адресована менеджерам и в которой написано, что им нужно делать.
Сумант Кумар, директор по развитию (инжиниринг), Innovative Business Solutions Organization, SAP

Мне понравилось обсуждение мотивации отдельных людей и команд наряду с лидерскими качествами, которые помогают в любой среде. Мы часто принимаем человеческий фактор как должное и обращаем внимание только на процесс.
Деннис Рубсам, старший директор, Seagate

Для руководителей, переходящих с традиционных способов управления проектами и с трудом усваивающих концепции Agile, книга станет откровением.

Пол ван Хаген, архитектор платформ и менеджер по совершенствованию программного обеспечения, Shell Global Solutions International B.V.

Книга содержит ключевые выводы не только о том, как собрать эффективную команду Agile, но и также о том, как руководство организации должно относиться к командам разработчиков, чтобы достичь успеха.
Том Спитцер, вице-президент по инжинирингу, EC Wise

Очень нужное дополнение в стремительно меняющемся мире разработки программного обеспечения, где в совершенно разных отраслях остро возрастает необходимость в ускорении и повышении объемов поставок продукции.
Кеннет Лю, старший директор по управлению программами, Symantec

Книга представляет собой всеобъемлющий, охватывающий все стороны руководства в Agile, справочник для менеджеров проектов, в которых уже применяется Agile, желающих улучшить свои показатели, или для руководителей, внедряющих Agile.
Брэд Мур, вице-президент по инжинирингу, Quartet Health

Весьма полезный сборник принципов, которые зарекомендовали себя в повышении уровня команд, работающих с учетом Agile-методов. Тут много реального ценного опыта, а не просто информация, собранная в один ресурс.
Дьюи Хоу, вице-президент по разработке продуктов, TechSmith Corporation

«Еще более эффективный Agile» — это отличное зеркало для внедрения Agile, в котором можно увидеть как положительные, так и отрицательные стороны.
Мэтт Шоутен, старший директор по развитию продуктов, Herzog Technologies

Сотрудники большинства компаний, вероятно, считают, что ведут разработку по Agile-методам, но они могут не обращать внимания на многие ключевые компоненты, позволяющие улучшить процесс разработки. Макконнелл опирается на исследования в области разработки ПО и собственный опыт в компании *Construx* и сводит эти знания в один компактный ресурс.
Стив Перрен, старший менеджер по развитию, Zillow

В книге рассматривают многие проблемы, с которыми мы долгие годы боролись, — жаль, не было этой книги, когда мы только начинали свой путь. Разделы «Рекомендации» просто великолепны.

Барри Сэйлор, вице-президент по разработке программного обеспечения, Micro Encoder Inc.

«Еще более эффективный Agile» — это кульминация 20-летнего опыта внедрения Agile. Так же как «Совершенный код» стал исчерпывающим руководством для разработчиков ПО в 1990-х, «Еще более эффективный Agile» станет исчерпывающим руководством для менеджеров в предстоящем десятилетии.
Том Керр, менеджер по разработке встраиваемого программного обеспечения, ZOLL Medical

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу **comp@piter.com** (издательство «Питер», компьютерная редакция).
Мы будем рады узнать ваше мнение!

Часть I. Введение в еще более эффективный Agile

В части I описаны фундаментальные концепции методологии гибкой разработки программного обеспечения под названием *Agile*. В частях II–IV мы углубляемся в конкретные предложения.
На концепции, представленные в части I, мы будем ссылаться на протяжении всей книги, поэтому если вы сразу начинаете с частей II–IV, имейте в виду, что весь этот материал основывается на идеях, изложенных в части I.
Если вы хотите начать с общего обзора, то перейдите к части V и прочитайте разделы «Наслаждайтесь плодами своих трудов» и «Краткое изложение ключевых принципов».

Глава 1. Введение

В начале 2000-х гг. ведущие специалисты в области программного обеспечения (ПО) подняли важный вопрос, касающийся методологии гибкой разработки Agile. Они обдумывали, может ли Agile обеспечивать качество, предсказуемость, выполнение крупных проектов, постепенные улучшения и работу в регулируемых отраслях. В то время их обеспокоенность была вполне обоснованна: первоначальные ожидания от Agile оказались завышены, внедрение методологии часто завершалось разочарованием, а достижение результатов нередко занимало больше времени, чем планировалось.

Отрасли разработки ПО нужны были время и опыт для того, чтобы на заре Agile отличить неудачи от подлинных достижений. В последующие годы специалисты в сфере разработки ПО улучшили некоторые из ранних методов, добавили новые и научились избегать неэффективных. В наши

дни применение современных Agile-методов предполагает улучшение качества, предсказуемости, продуктивности и эффективности одновременно.

Более чем двадцать лет моя компания *Construx Software*сотрудничала с организациями, которые занимались разработкой ПО, начиная с мобильных игр и заканчивая программами для медицинского оборудования. Мы помогли преуспеть сотням организаций с помощью метода «последовательной разработки» и на протяжении последних 15 лет получаем все лучшие результаты от применения Agile. Мы видели, как организациям удавалось значительно сократить время выполнения производственных циклов, улучшить качество, оперативность в поддержке клиентов и увеличить прозрачность посредством применения Agile.

Бˆольшая часть литературы по Agile ориентирована на амбициозные компании с высокими темпами роста на новых рынках, например на *Netflix*, *Amazon*, *Etsy*, *Spotify* и т.п. Но как быть, если ваша компания занимается не таким передовым софтом? Что насчет компаний, которые производят ПО для научных приборов, оргтехники, медицинского оборудования, бытовой электроники, тяжелого машиностроения или аппаратуры для реализации технологических процессов? Мы обнаружили, что современные Agile-методы, если применять их с упором на то, что лучше подходит для определенного бизнеса, предоставляют преимущества и для таких программных продуктов.

Почему важен эффективный Agile

Компании хотят большей эффективности разработки ПО для своего же блага. Они также хотят большей эффективности, потому что от ПО зависят многие другие аспекты бизнеса. В докладе State of DevOps говорится, что «компании с высокопроизводительной организацией ИТ имели в два раза больше шансов превзойти свои показатели прибыльности, доли рынка и производительности» [Puppet Labs, 2014]. У высокопроизводительных компаний было в два раза больше шансов достичь или превзойти свои цели по удовлетворенности клиентов, качеству и количеству работ, эффективности и другим задачам.

Выборочное применение современных Agile-методов при должном информировании предлагает проверенный путь к большей эффективности разработки ПО и все преимущества, которые можно от этого получить.

К сожалению, большинство организаций не осознают потенциал Agile-методов, потому что большинство реализаций Agile-методов неэффективно. Например, Скрам (Scrum), наиболее часто встречающийся гибкий метод управления проектами, может быть невероятно мощным инструментом, но чаще мы видели такие реализации, которые сводили на нет все преимущества. В графике, приведенном ниже, дано сравнение

средней команды, работающей по Скраму, как это часто бывает, и команды, работающей по Скраму правильно.

Обычно мы видим только одну из ключевых составляющих Скрам, которую применяют эффективно (дейли-скрам, стендап), и даже такое не повсеместно. Некоторые элементы Скрама применяются лишь время от времени или не применяются вовсе. (Цифры из этого графика подробно описаны в главе 4.) Плохая реализация потенциально хорошего метода — не единственная причина провала Agile. Методология Agile стала общим понятием, охватывающим тьму методов, принципов и теорий. Мы видели, как не получается внедрить Agile, потому что организация не совсем понимала, что собой представляет Agile.

В целом, в Agile есть более и менее эффективные методы, поэтому мы видели неудачи некоторых организаций из-за того, что они ошиблись с выбором методов.

Организации могут добиться куда лучшей производительности, и в этой книге рассказано, как ее достичь.

Для кого эта книга

Эта книга для руководителей высшего звена, вице-президентов, директоров, менеджеров, тимлидов и прочих руководителей организаций, которые хотят эффективно внедрить Agile. Если у вас есть техническая квалификация, но недостаточно опыта в применении современных Agile-методов, эта книга для вас. Если у вас нет технической квалификации и вы просто хотите узнать о рабочих практических методах Agile, то смело читайте дальше (можно пропустить специализированную часть).

Если же вы многое узнали о методах Agile 10–15 лет назад, но с тех пор не обновляли свои знания и ваш Agile-подход устарел, то и в этом случае эта книга для вас.

И самое главное — если ваша компания уже пыталась внедрить аспекты Agile, но вы недовольны результатом, читайте дальше!

Чем она отличается от прочих книг по Agile

Эта книга не о том, как работать по Agile «правильно», она о том, как извлечь максимально возможную пользу из тех Agile-методов, которые разумно применять в вашем бизнесе.

Я затрагиваю темы, важные в мире бизнеса, но на которые не обращают внимания борцы за чистоту Agile: общие вызовы при внедрении Agile, как внедрить его только в отдельной части организации, предсказуемость при использовании Agile, лучшие способы его применения в географически разобщенных командах, применение Agile в регулируемых отраслях. И это только несколько тем, которые раскрыты в этой книге.

Большинство книг по Agile писали ИТ-евангелисты. Они либо выступают за какие-то конкретные Agile-методы, либо агитируют за Agile в целом. Я не евангелист Agile. Я сторонник того, что работает, и противник того, что много обещает, но не приносит результатов. В этой книге методология Agile представлена не как движение, которое требует повышенной сознательности, а как набор специальных управленческих и технических методов, эффект и взаимодействие которых можно понять и со стороны бизнеса, и со стороны производства.

Я бы не смог написать эту книгу в начале 2000-х гг., потому что тогда мир разработки ПО еще не вобрал в себя практический опыт применения Agile, чтобы можно было с какой-либо уверенностью судить, что работает, а что нет. На сегодняшний день мы уже знаем, что некоторые из методов, которые популяризировали в наибольшей мере, не доказали свою эффективность. В то же время другие методы, которые были не так известны, впоследствии себя проявили как надежные рабочие лошадки для эффективного внедрения Agile. В этой книге приводится их разбор.

Энтузиасты Agile могут раскритиковать эту книгу за то, что она якобы не относится к передовой литературе по Agile. Но в этом и смысл: в книге делается акцент на практических методах, доказавших свою эффективность. История Agile полна идей, которые удалось успешно реализовать паре энтузиастов в некоторых организациях, но которые в целом не были признаны полезными. Эта книга не останавливается на таких методах с ограниченным применением.

«Еще более эффективный Agile» представляет собой путеводитель в мире современных функционирующих Agile-методов, а также содержит несколько предостережений относительно того, что некоторых методов и идей нужно избегать. Эта книга не учебник по Agile, а руководство, призванное помочь управленцам в сфере разработки ПО вычленять сигнал из шума.

Структура

Эта книга начинается с *предыстории и объяснения контекста*, потом переходит к *сотрудникам компаний и командам*, затем к *рабочим практическим методам*, которые используются отдельными сотрудниками компаний и командами, далее к *организациям*, внутри которых команды применяли рабочие методы на практике, и, наконец, к *подведению итогов и перспективам*.

Вступления в начале каждой части книги содержат краткое описание, чтобы вы могли решить, нужно ли вам читать ту или иную часть и в каком порядке.

А что думаете вы?

Я бы не написал эту книгу без огромного количества отзывов. Исходную рукопись тщательно проверяла моя команда из Construx Software. Я попросил сторонних добровольцев прорецензировать черновик и более 300 руководителей в отрасли разработки ПО оставили свыше 10 тысяч комментариев. Их неоценимая помощь принесла моей книге огромную пользу.

Как вы отреагировали на эту книгу? Совпадает ли материал с вашим опытом? Помогла ли книга в решении каких-нибудь проблем? Буду рад видеть ваши сообщения. Контакты для связи со мной указаны ниже.

Белвью, штат Вашингтон (США) 4 июля 2019 года

stevemcc@construx.com
Linkedin.com/in/stevemcc
SteveMcConnellConstrux
@Stevemconstrux
MoreEffectiveAgile.com

Глава 2. В чем отличие Agile?

Большинство книг по Agile, в которых есть глава вроде «В чем заключается отличие Agile?», сразу же погружают нас в описание Манифеста Agile 2001 года и связанных с ним методов, которым уже 20 лет.

Эти документы служили важным и полезным целям 20 лет назад, но с того времени Agile-методы постоянно развивались, и ни один из этих исторических источников не может точно характеризовать самые ценные стороны Agile сегодня.

Так чем же отличается Agile сегодня? Исторически движение Agile противопоставлялось каскадной модели разработки. Претензия к ней заключалась в том, что такая модель вынуждала заранее выполнять планирование на 100%, готовить требования на 100%, заранее выполнять

проектирование на 100% и так далее. Это была точная характеристика разработки, которая велась по каскадной модели, однако в реальности так работать никогда не удавалось. Существовали разные способы *поэтапного проектирования*.

Разработка по каскадной модели в ее чистом виде велась главным образом в ранних проектах Министерства обороны США, и ко времени написания Манифеста Agile эта очень сырая реализация уже была вытеснена более сложными моделями [1].

В наибольшем противопоставлении с Agile сейчас находится модель последовательной разработки. Во избежание путаницы сравнение представлено в табл. 2.1.

Короткие и долгие циклы релиза

Команды, применяющие Agile-методы, разрабатывают рабочее ПО по циклам, которые длятся дни или недели. У команд, применяющих модель последовательной разработки, циклы измеряются месяцами или кварталами.

Таблица 2.1. Различия между моделью последовательной разработки и Agile

Последовательная модель	Agile
Долгие циклы релиза	Короткие циклы релиза
Большая часть сквозной разработки ведется большими партиями в течение долгих циклов релиза	Большая часть сквозной разработки ведется малыми партиями за короткие циклы релиза
Подробное заблаговременное планирование	Обобщенное заблаговременное планирование вместе со своевременным подробным планированием
Подробные заблаговременные требования	Обобщенные заблаговременные требования вместе со своевременными подробными требованиями
Заблаговременное проектирование	Стихийное проектирование
Тестирование в конце, часто как отдельная задача	Непрерывное, автоматизированное тестирование, интегрированное в разработку
Нечастое упорядоченное сотрудничество	Частое упорядоченное сотрудничество
В целом, подход идеалистичен, ориентирован на контроль и заранее намеченный план	В целом, подход эмпиричен, ориентирован на оперативность и совершенствование

Сквозная разработка ведется маленькими и большими партиями

Agile делает упор на глубокую разработку, которая включает в себя подробные требования, проектирование, написание кода, тестирование и написание документации малыми партиями, то есть за промежуток времени реализуется небольшое количество функций или требований. При последовательной модели разработки упор делают на то, что все требования проекта, все проектирование, написание кода и тестирование проходят через конвейер разработки одновременно большими партиями.

Своевременное и заблаговременное планирование

Когда применяют Agile, обычно проводят небольшое заблаговременное планирование, а большая часть подробного планирования проходит по мере необходимости.

При хорошо организованной последовательной модели разработки весомая часть планирования также проводится по мере необходимости, но методы этой модели, например метод освоенного объема проекта, обращают повышенное внимание на более подробное планирование заранее.

Своевременные и заблаговременные требования

В разработке по Agile стараются как можно меньше работать с требованиями заранее (большее внимание уделяется самой сути проекта, чем деталям). Предполагается откладывание подавляющего большинства подробных требований до тех пор, пока в ходе проекта не понадобится их реализация. Последовательная модель разработки разбирает большую часть подробных требований заранее.

Требования — это та область, в которой современные Agile-методы вышли за рамки идей, сопутствующих методологии Agile в начале 2000-х. Я рассмотрю эти изменения в главе 13 «Создание требований еще более эффективного Agile» и в главе 14 «Определение приоритетов требований еще более эффективного Agile».

Стихийное и заблаговременное проектирование

Как и в случае с планированием и требованиями, Agile откладывает подробную проработку проектирования до тех пор, пока она не понадобится, с минимальным вниманием к заблаговременному построению архитектуры. Последовательная модель разработки делает упор на более глубокую и подробную проработку заранее.

Признание пользы заблаговременного проектирования и разработки архитектуры в некоторых случаях — еще одна область, в которой Agile вышел за пределы своей ранней философии 2000-х годов.

Непрерывное, автоматизированное тестирование, интегрированное в разработку, и отдельный тест в конце

Методология Agile настаивает на том, что тестирование — это что-то, что делается параллельно с написанием кода, а иногда даже и до его написания. Оно проводится интегрированными командами, которые включают как разработчиков, так и специалистов по тестированию. Последовательная модель разработки рассматривает тестирование как деятельность, которая ведется отдельно от разработки и, как правило, уже

по ее окончании. Agile делает упор на автоматизацию тестов, благодаря чему их может проводить большее количество людей с большей частотой.

Частое упорядоченное и нечастое упорядоченное сотрудничество

Agile подчеркивает важность частого упорядоченного сотрудничества. Такая совместная работа зачастую длится недолго (пятнадцатиминутные стендапы), но она упорядочена в ежедневном и еженедельном рабочем ритме Agile. Последовательная модель разработки, конечно, не препятствует сотрудничеству, но в то же время особо его не поощряет.

Эмпиричность, оперативность, ориентация на совершенствование и идеалистичность, намеченный план, ориентация на контроль

Команды, работающие по Agile, делают акцент на эмпирическом подходе. Они сосредоточены на получении информации из окружающей среды. Команды, работающие по модели последовательной разработки, также получают информацию исходя из реального опыта, но они делают больший упор на разработку плана, пытаясь подчинить реальность своему порядку, вместо того чтобы наблюдать за происходящим и постоянно к ней адаптироваться.

Что общего между методологией Agile и моделью последовательной разработки

Сравнения подходов Agile и модели последовательной разработки обычно направлены на то, чтобы показать преимущества Agile перед этой моделью или наоборот. Это нечестно и бессмысленно. Хорошо управляемые проекты прежде всего обращают внимание на высокий уровень менеджмента, взаимодействия с клиентами, требований, проектирования, написания кода и тестирования, независимо от того, какой подход используется, Agile или последовательная разработка.

Последовательная разработка при наилучшей организации работает неплохо. Однако если вы изучите различия, приведенные в табл. 2.1, и мысленно перенесете их на свои проекты, то сможете увидеть некоторые намеки на то, почему Agile проявляет себя во многих случаях лучше, чем модель последовательной разработки.

Источник преимуществ Agile

Преимущества методологии Agile не возникают из мистического применения понятия «Agile», но являются следствием легко объяснимых

эффектов тех свойств, на которые делает упор Agile и которые приведены в табл. 2.1.

Короткие циклы релиза позволяют устранять дефекты быстрее и дешевле, меньше времени вкладывается в разработку тупиковых решений, обеспечиваются более отзывчивая поддержка клиентов, более быстрое исправление курса и более короткие пути к повышению доходов и сокращению эксплуатационных расходов.

Сквозная разработка малыми партиями дает преимущества по схожим причинам — более тесная обратная связь позволяет быстрее находить и исправлять ошибки с меньшими затратами.

Своевременное планирование в результате позволяет затрачивать меньше времени на создание подробных планов, которые позже игнорируются или отбрасываются.

*Своевременные требования*позволяют затрачивать меньше усилий на предварительные требования, которые при изменениях в конечном итоге отбрасываются.

В результате *стихийного проектирования* меньше труда вкладывается в заблаговременные проектные решения для требований, меняющиеся впоследствии, не говоря уже о решениях, которые работают не так, как было запланировано. Заблаговременное проектирование может быть эффективно, но применительно к спорным требованиям оно подвержено ошибкам и затратно.

Непрерывное автоматизированное тестирование, интегрированное в процесс разработки, дает более тесную обратную связь между временем, когда недочет появился, и временем, когда его обнаружили, что способствует снижению стоимости исправления недочетов и усиливает внимание на изначальном качестве кода.

Частое упорядоченное сотрудничество снижает вероятность ошибок, возникших в результате недопонимания, которые могут значительно способствовать появлению недочетов в требованиях, проектировании и прочих видах деятельности.

*Акцент на модели эмпиричного и оперативного совершенствования*позволяет командам быстрее учиться на своем опыте и со временем улучшать продукт.

Разные организации акцентируют внимание на разных сторонах Agile. Организация, которая разрабатывает ПО, где безопасность является ключевым моментом, как правило, не станет применять стихийное проектирование. Стихийное проектирование может сэкономить деньги, но соображения безопасности важнее.

Похожим образом организация, которая несет высокие расходы при каждом выпуске ПО, скажем, если софт встроен в труднодоступное устройство либо если приходится следовать нормативам, не станет часто

выпускать релизы. Обратная связь, полученная от частых релизов, может сэкономить деньги некоторым организациям, но другим организациям она может обойтись дороже сэкономленных средств.

Когда вы выходите за рамки мышления, где Agile — это неделимая концепция, которую нужно применять либо полностью, либо никак, вы можете применять Agile-методы по отдельности. Вы начинаете осознавать, что некоторые Agile-методы полезнее других, а некоторые из них полезнее при ваших особых обстоятельствах. Если вашей организации нужно поддерживать гибкость в бизнесе, современные Agile-методы вам подойдут. Если вашей организации нужно поддерживать уровень качества, предсказуемости, производительности и остальных характеристик, неявно свойственных Agile, современные Agile-методы также представляют ценность.

Граница Agile

Большинство организаций не могут полностью перейти на Agile. Ваша компания, возможно, не видит пользы от применения Agile в управлении кадрами или закупках. Даже если по Agile будет работать вся организация, можно обнаружить, что ваши клиенты или поставщики не соответствуют Agile в той же мере, что и вы.

Полезно понимать, где находится граница между той частью компании, которая работает по Agile, и той, в которой Agile не используется, при этом определять текущую и желаемую границу.

Если вы менеджер высшего звена, граница Agile может отделять вашу организацию полностью. Если вы ведущий технический руководитель, граница может отделять все техническое подразделение. Если вы менеджер более низкого звена, то граница может отделять только ваши собственные команды. На рис. 2.1 дан пример.

Рис. 2.1. Пример границы Agile. Здесь применение Agile-методов ограничено только техническим подразделением

Непонимание границы Agile может привести к неоправданным ожиданиям и прочим проблемам. Рассмотрим такие ситуации:

- Agile в разработке ПО и нормативы, не соответствующие Agile.
- Agile в продажах и разработка ПО не по Agile.
- Agile в разработке ПО и корпоративные клиенты, не работающие по Agile.

В каждой организации есть граница. Как сильно вы хотите внедрить Agile в своей организации? Что лучше всего поможет вашему бизнесу?

Рекомендации

Изучайте

- Подумайте о том, в какой степени вы раньше были уверены, что Agile должен применяться либо полностью, либо никак. В какой степени это повлияло на ваш подход к совершенствованию управления и технических методов?
- Поговорите по меньшей мере с тремя техническими руководителями из вашей организации. Спросите у них о том, что они понимают под словом «Agile». Спросите их, какие именно методы они имеют в виду. В какой степени взгляды технических руководителей на Agile совпадают? Есть ли у них согласие в том, что не является Agile?
- Поговорите с нетехническими руководителями о том, что для них Agile. Как они воспринимают границу или контакт между их работой и работой команд разработчиков?
- Пересмотрите свое портфолио проектов с точки зрения различий, приведенных в табл. 2.1. Оцените ваши проекты по каждому пункту, где 1 — это полностью последовательная разработка, а 5 — полностью Agile.

Приспосабливайтесь

- Запишите предварительный план проведения границы Agile в своей компании.
- Создайте список вопросов, на которые надеетесь получить ответ по мере прочтения этой книги.

Дополнительные ресурсы

[Эндрю Стеллман и Дженнифер Грин, 2013]. Learning Agile: Understanding Scrum, XP, Lean, and Kanban. Эта книга — хорошее введение в концепцию Agile с точки зрения профессионалов Agile.

[Бертран Мейер, 2014]. Agile! The Good, the Hype and the Ugly. Эта книга начинается с развлекательной критики излишеств в движении Agile и определяет наиболее полезные принципы и методы, связанные с движением Agile.

Глава 3. Отвечая на вызовы сложности и неопределенности

При создании проектов разработки ПО специалисты долгое время бились над вопросом, как справиться со сложностью, которая являлась источником многих вызовов, среди которых низкое качество, опоздание и откровенный провал.

В этой главе представлен набор методов для понимания неопределенности и сложности, известный как Cynefin (произносится как «киневин»). В ней рассказано, как Cynefin можно применить к модели последовательной разработки и Agile. Затем в этой главе представлена модель принятия решений в условиях неопределенности и сложности, известная как петля Бойда, или цикл НОРД (наблюдение, ориентация, решение, действие). Приведены случаи, в которых принятие решений по методу петли Бойда имеет преимущества над типичными подходами к принятию решений в модели последовательной разработки.

Cynefin

Набор методов Cynefin был создан Дэвидом Сноуденом, работавшим в *IBM*, в конце 1990-х [Курц, 2003].

С тех пор развитие этого метода продолжается Сноуденом и другими специалистами [Сноуден, 2007]. Сноуден описывает Cynefin как «осмысленный набор методов». Он помогает понять, какая тактика будет полезна в зависимости от сложности и неопределенности ситуации.

Набор методов Cynefin состоит из пяти контекстов. Каждый из них имеет собственные свойства и предлагаемые ответы. Контексты изображены на рис. 3.1.

Cynefin — это валлийское слово, которое означает «среда обитания», или «окрестности». Контексты не считаются категориями, скорее их рассматривают как кластеры значений, которые являются источником свойства, отраженного в названии, — «срˆеды обитания».

Рис. 3.1. Набор методов Cynefin — это полезный и осмысленный инструмент, который можно применять в отрасли разработки ПО

Контексты «Сложные» и «Запутанные» наиболее актуальны в области разработки ПО. Все пять контекстов описаны в следующих разделах с целью прояснения.

Контекст «Очевидные»

При контексте «Очевидные» (Obvious) проблемы хорошо поддаются пониманию, а решения приходят на ум сами собой. Буквально все сходятся на одном и том же правильном решении. Связь между причиной и следствием проста и прямолинейна. Это сфера применения паттерна: программируемое процедурное механическое поведение.

Подход к проблемам контекста «Очевидные» в наборе методов объясняется следующим образом:
осознайте.классифицируйте.реагируйте
Приведем несколько примеров проблем из контекста «Очевидные»:

- Прием заказа в ресторане.
- Обработка платежа по кредиту.
- Наполнение бака автомобиля топливом.

На более глубоком уровне команды разработчиков сталкиваются с большим количеством «очевидных» проблем, например: «Этот оператор if должен проверять на наличие 0, а не 1».

На уровне проекта сложно найти примеры «очевидных проблем» вроде описанных в Cynefin. Когда вы в последний раз в области разработки ПО видели проблему какого угодно размера, у которой было бы только одно верное решение, с которым были бы все согласны? Существует хорошее исследование в области разработки ПО, которое показывает, что когда разным проектировщикам встречается одна и та же проблема, они создают

решения, объем кода которых превышает объем кода, необходимый для выполнения задач проектирования, в десять раз [Макконнел, 2004]. По моему опыту, эта разница существует даже в выполнении такой, казалось бы, простой задачи, как «создать краткий отчет». «Одно правильное решение» на практике оказывается сколь угодно разным. Так что, думаю, что за рамками программ уровня «hello world» в разработке ПО не бывает «очевидных» проблем. Пока мы рассматриваем разработку ПО в крупных масштабах, полагаю, что можно спокойно игнорировать контекст «Очевидные».

Контекст «Сложные»

При контексте «Сложные» (Complicated) проблема известна вам в общих чертах, вы знаете, какие вопросы нужно задать и как получить на них ответы.

Есть много правильных вариантов ответа. Связь между причиной и следствием сложна, и приходится анализировать, изучать и применять экспертные знания, чтобы понять связь между причиной и следствием. Не все видят или понимают причинно-следственную связь, что делает «сложный» контекст уделом экспертов.

Подход к проблемам контекста «Сложные» в наборе методов объясняется следующим образом:
осознайте.анализируйте.реагируйте
Этот подход противопоставлен подходу контекста «Очевидные» в том, что средний этап требует анализа, а не простой классификации проблемы и выбора одного правильного ответа.

Приведем несколько примеров проблем из контекста «Сложные»:

- Определение причины стука двигателя.
- Приготовление изысканного блюда.
- Написание запроса к базе данных для получения конкретного результата.
- Устранение в производственной системе ошибки, которая приводит к неполному обновлению.
- Определение приоритетности требований пользователей для версии 4.1 зрелой системы.

Все эти примеры обладают одной и той же чертой — сначала нужно сформулировать понимание проблемы и подход к ней, затем найти прямой путь решения проблемы и перейти к ее решению.

Большое количество разработок и проектов в области создания ПО попадают в контекст «Сложные». Исторически контекст относился к сфере влияния модели последовательной разработки.

Если проект главным образом находится в контексте «Сложные», может сработать линейный, последовательный подход, строго полагающийся на

заблаговременное планирование и анализ. При таком подходе сложности возникают тогда, когда проблему трудно ясно определить.

Контекст «Запутанные»

Определяющей характеристикой контекста «Запутанные» (Complex) является то, что связь между причиной и следствием не сразу очевидна даже для экспертов. В отличие от контекста «Сложные», вы не знаете всех вопросов, которые нужно задать, поэтому часть испытания — раскрытие вопросов. Никакой предварительный анализ не решит проблему, и для продвижения к решению необходимы эксперименты. Сколько-то неудач — это часть процесса, а решения часто приходится принимать на основании неполных данных.

У «запутанных» проблем связь между причиной и следствием можно понять только задним числом — некоторые элементы проблемы появляются стихийно. Тем не менее при достаточном количестве и качестве экспериментов связь между причиной и следствием становится достаточно понятной, для того чтобы принять адекватное решение. Сноуден считает, что «запутанные» проблемы относятся к области сотрудничества и терпения, нужно не мешать решениям самим приходить на ум.

Рекомендуемый подход к проблемам контекста «Запутанные» в Cynefin объясняется следующим образом:
исследуйте.осознайте.реагируйте
Этот подход противопоставлен решению «сложных» проблем в том, что вы не сможете проанализировать, как выйти из положения. Сначала нужно исследовать. В конечном итоге, анализ станет актуальным, но не сразу.

Приведем несколько примеров проблем из контекста «Запутанные»:

- Покупка подарка кому-то, кому трудно угодить, — вы вручаете подарок и знаете, что вам придется его обменять!
- Исправление ошибки производственной системы, в которой средства диагностики вызывают исчезновение ошибки во время отладки, но не во время эксплуатации.
- Выявление пользовательских требований для совершенно новой системы.
- Создание ПО, работающего на базовом оборудовании, которое все еще дорабатывается.
- Обновление ПО, в то время как конкуренты обновляют свое.

Большое количество разработок и проектов в области создания ПО попадают в контекст «Сложные», а это владения Agile и итеративной разработки. Если проект в основном находится в контексте «Сложные», рабочий подход заключается в том, чтобы сначала экспериментировать и разведывать, прежде чем проблема станет полностью определенной в принципе.

На мой взгляд, провал последовательной разработки в применении к «запутанным» проектам в значительной мере породил методологию Agile.

В некоторых случаях получается исследовать «запутанный» проект достаточно подробно для того, чтобы превратить его в «сложный». Остальную часть проекта можно вести с помощью подходов, которые годятся для «сложных» проектов. В других случаях «запутанный» проект сохраняет значительное количество «запутанных» составляющих на протяжении всего жизненного цикла проекта. Усилия, затраченные на конвертацию проекта в «сложный», занимают время, которое лучше потратить на выполнение проекта с использованием подходов, соответствующих «запутанным» проектам.

Контекст «Хаотические»

Контекст «Хаотические» (Chaotic) немного отстоит от паттерна, который можно ожидать на основании сведений о трех предыдущих контекстах. При контексте «Хаотические» связь между причиной и следствием бурная и подвижная. Невозможно обнаружить связь между причиной и следствием даже после повторных экспериментов, даже задним числом.

Вы не знаете, какие вопросы нужно задавать, а разведка и эксперименты не дают единообразных ответов.

Контекст также включает в себя дефицит времени, которого нет в остальных контекстах.

Cynefin характеризует контекст «Хаотические» как контекст решительного лидерства, ориентированного на действие. Рекомендованный подход — навести порядок в хаосе и сделать это как можно скорее:
действуйте.осознайте.реагируйте
Приведем несколько примеров проблем из контекста «Хаотические»:

- Обеспечение помощи во время стихийных бедствий, в то время как бедствие еще происходит.
- Прекращение кидания едой в школьной столовой.
- Исправление ошибки в производственной системе путем отката к предыдущей версии, поскольку никакие анализ и разведка не помогли обнаружить причину ошибки.
- Определение набора функций в напряженной среде, где требования возникают и меняются в силу амбиций влиятельных стейкхолдеров.

Найти пример «хаотической» проблемы в размерах проекта в отрасли разработки ПО затруднительно, если не невозможно. Пример исправления ошибки имеет вид «нет времени анализировать, надо действовать», но это не пример в размерах проекта.

Пример набора функций — это пример в размерах проекта, но в этом случае нет острой составляющей давления сроков, поэтому можно сделать

вывод, что этот пример нельзя отнести к «хаотической» проблеме с точки зрения Cynefin.

Контекст «Беспорядочные»

В середине диаграммы Cynefin расположился контекст «Беспорядочные» (Disorder), в котором нет ясного представления о том, к какому контексту относится ваша проблема. Подход, рекомендованный Cynefin к решению проблем контекста «Беспорядочные», таков: разобрать проблему по составляющим, а затем определить, к какому контексту какая составляющая относится.

Cynefin предлагает способ определения различных составляющих и соответствующего обращения с ними. Вы подходите к требованиям, проектированию и планированию работы одним способом в решении проблем из контекста «Запутанные» и другим способом в решении «сложных» проблем. Невозможно найти универсальный подход, работающий везде.

Большая часть проблем в размерах проекта разработки ПО не всегда легко укладывается в рамки одного контекста; имейте в виду, что все эти контексты находятся по соседству и представляют собой совокупности значений, которые находятся в одном кластере. Разные составляющие проблемы или системы могут проявить разные свойства, некоторые могут относиться к «сложным», в то время как другие — к «запутанным».

Cynefin и вызовы в области разработки

Cynefin — это интересный, полезный и осмысленный набор методов, а все пять контекстов можно распространить за пределы проблем в области разработки ПО. Но, как показано на рис. 3.2, контексты «Хаотические» и «Очевидные» нельзя распространить на уровень целого проекта по причинам, которым я привел выше. Это означает, что для практических целей проекты разработки ПО должны ориентироваться в основном на контексты «Сложные», «Запутанные» и «Беспорядочные» (при этом «Беспорядочные» нужно привести в порядок и определить, относятся ли проблемы к «сложным», «запутанным» или сразу к обоим типам).

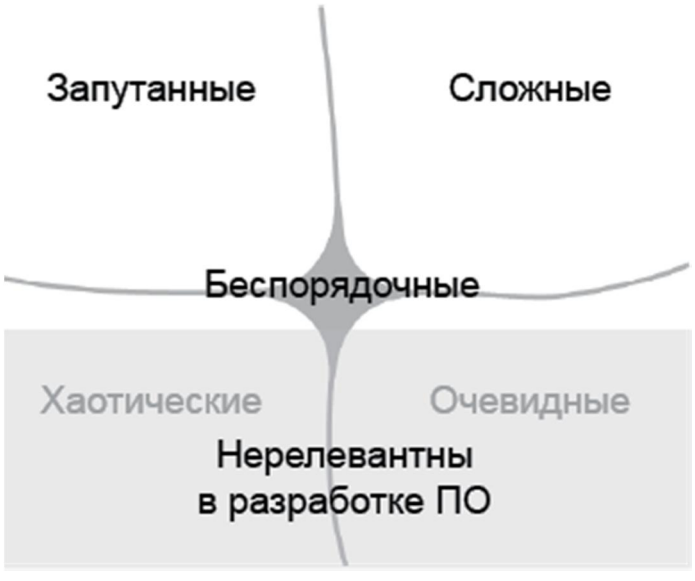


Рис. 3.2. Связь между контекстами Cynefin и вызовами в отрасли разработки ПО

Принимая во внимание то, что у вас есть на выбор только два контекста Cynefin, возникает вопрос: «А что делать, если я прогадал и определил не тот контекст?»

Как показано на рис. 3.3, чем больше неопределенности в проекте, тем лучше срабатывает подход контекста «Запутанные» (Agile), чем контекста «Сложные» (последовательная разработка).

Рис. 3.3. Сравнение вероятности успешного решения разных видов проблем с помощью последовательной разработки и Agile

Если вы думаете, что ваш проект по большей части относится к «запутанным», а потом он оказывается именно «сложным», вы зря потратите время на исследование и эксперименты. Если посмотреть поверхностно, то в этом случае вас наказывают за отнесение проекта не к тому контексту тем, что проект якобы теряет в эффективности. Но это можно оспорить, ведь эксперименты, проведенные вами, наверняка углубили понимание вами проекта и улучшили ваш подход к нему.

Если вы думаете, что ваш проект относится к «сложным», а он оказывается скорее «запутанным», придется потратить время на анализ, планирование и по крайней мере частичное выполнение проекта, который вы поняли не так хорошо, как думали сначала. Если вы находитесь на первом месяце полугодового проекта и понимаете, что идете куда-то не туда, возможно, необходима полная перезагрузка проекта. Если идет пятый месяц полугодового проекта, проект можно полностью сворачивать.

Последствия неправильного отнесения проекта к «запутанным» не так велики, как неправильного отнесения к «сложным». Таким образом, деньги надежно расходуются на работу с проектом как с «запутанным» с применением Agile-методов, если только вы не полностью уверены в том, что проект относится к «сложным», в случае с которым может сработать подход последовательной разработки.

Успех в «запутанных» проектах: петля Бойда

Полезная модель для решения проблем «запутанных» проектов — это петля Бойда, она же петля НОРД (OODA). Как показано на рис. 3.4, «НОРД» означает «наблюдение, ориентация, решение, действие», иногда петлю еще называют «циклом НОРД».

Рис. 3.4. Основу петли Бойда составляют четыре этапа, первый из которых — наблюдение

Петля Бойда была разработана полковником ВВС США Джоном Бойдом, поскольку его разочаровывали результаты боев, в которых участвовали воздушные силы. Он изобрел петлю НОРД как способ ускорить принятие решений, чтобы делать это быстрее противника, тем самым предвосхитить его решения. Петля Бойда — это методический подход к установлению контекста, формулированию плана, выполнению работы, наблюдению результатов и включению полученных знаний в следующий цикл.

Наблюдение

Петля Бойда начинается с этапа «Наблюдение». Наблюдайте за происходящей ситуацией, за пределами актуальной информации, за ситуацией со всех сторон, которые проявляют себя (стихийно), затем наблюдайте, как проявившие себя стороны ситуации взаимодействуют с внешней средой. Поскольку метод петли Бойда уделяет большое внимание наблюдению, можно сделать вывод, что это — эмпирический подход, основанный на наблюдении и получении опыта.

Ориентация

На этапе «Ориентация» вы соотносите наблюдения с ситуацией. Бойд заявил, что мы соотносим их с нашим «генетическим наследием, культурной традицией, усвоенным опытом и возникающими обстоятельствами» [Адольф, 2006]. Проще говоря, вы решаете, какая информация имеет значение для вас, а затем выбираете, как реагировать.

Этап «Ориентация» позволяет оспорить ваши предположения, приспособиться к культурным и корпоративным слепым пятнам и, в общем, сделать данные и сведения непредвзятыми. По мере того как вы

ориентируетесь, вы переключаетесь между приоритетами, основываясь на улучшении понимания ситуации. Это позволяет осознавать значение деталей, на которые другие не обратили внимания. Классический пример — iPhone от компании *Apple*. Другие игроки придавали первостепенное значение мегапикселям, качеству радиосигнала и времени автономной работы. *Apple*же ориентировалась совершенно на другое, сосредоточившись на создании информационного мобильного устройства с прорывным пользовательским интерфейсом. iPhone уступал почти по всем характеристикам мобильного телефона в традиционном понимании, но это было не важно, потому что компания ориентировалась на решение другой проблемы, которая в итоге оказалась более важной.

Решение

На этапе «Решение» вы принимаете решение на основе вариантов, которые были определены на этапе «Ориентация». В контексте военных действий часто приходится принимать такое решение, которое нарушит планы соперника — это называется «пробивание петли Бойда соперника». Иногда это воспринимается как действия на опережение соперника, но если выражаться точнее — это действия в другом темпе.

В бейсболе питчер, делающий бросок с переменой скорости (медленная подача), когда отбивающий ожидает быстрой подачи, эффективно пробивает петлю Бойда соперника, действуя при этом медленнее. Другой способ подумать об этом — подчинить игру вашего соперника своим правилам (то, что сделала *Apple* на рынке с помощью iPhone).

Действие

И наконец, действуйте посредством реализации решения. И уже затем возвращайтесь обратно к этапу «Наблюдение», чтобы увидеть, какое влияние оказали ваши действия («возникающие обстоятельства»), и начинайте цикл снова.

За рамками основ петли Бойда

Несмотря на то что в своей основе петля Бойда кажется линейным циклом, полная петля Бойда позволяет осуществлять неявное руководство и контроль, как показано на рис. 3.5.

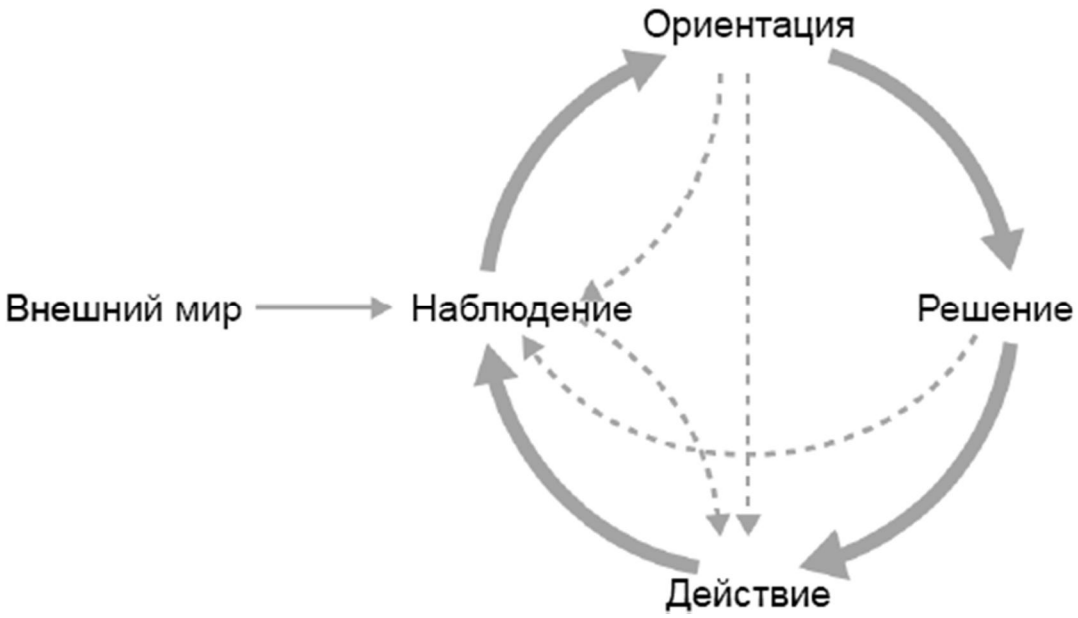


Рис. 3.5. Полная петля Бойда. С каждого этапа можно перейти сразу к этапу «Действие» или назад к «Наблюдению», как отмечено пунктирными линиями

Вам не нужно проходить полный цикл петли Бойда, чтобы отдернуть руку от горячей плиты, вы переходите сразу от наблюдения к действию. Если вы сталкиваетесь с распознанным паттерном на этапе «Наблюдение» или «Ориентация», вы можете сразу переходить к этапу «Действие» («Идет дождь, поэтому я лучше поеду на машине, чем пойду пешком»).

В то время как другой подход к принятию решений может потребовать от вас пройти полный цикл во имя педантичности, петля Бойда делает упор на быстроту принятия решений, благодаря чему вы можете переиграть соперника.

Петля Бойда и последовательная разработка

Проблемы в области разработки ПО, которые решают сегодня компании, имеют много междисциплинарных, стихийных характеристик, которые нельзя естественным образом решить с помощью подхода последовательной разработки. Agile-методы лучше подходят для решения таких проблем, эффективнее управляя рисками и обладая режимами смягчения неудач.

Ключевое различие между петлей Бойда и подходом последовательной разработки (приведено в табл. 3.1) в том, что петля Бойда обращает внимание прежде всего на наблюдение за окружающей средой и реакцией на ее проявления, тогда как подход модели последовательной разработки пытается контролировать проявления окружающей среды.

Таблица 3.1. Различия между подходами модели последовательной разработки и петли Бойда (Agile)

Последовательная разработка	Петля Бойда (Agile)
Изначальное внимание на планировании	Изначальное внимание на наблюдении
Упор на долгие сроки	Упор на короткие сроки
Предсказуемый	Оперативный
Идеалистичный	Эмпирический
Рассматривает неопределенность как риск	Рассматривает неопределенность как возможность
Ориентирован на контроль	Ориентирован на улучшение
Хорошо справляется со «сложными» проблемами	Хорошо справляется с «запутанными» проблемами, а также со «сложными»

Подход последовательной разработки требует большого количества заблаговременного планирования, проектирования и так далее. Петля Бойда (Agile) предлагает выполнять большую часть работы точно в срок, в том числе планирование с требованиями, проектирование и реализацию. При разработке по Agile не пытаются настолько предвидеть ситуацию, насколько это требуется при последовательной разработке.

Последовательная разработка может рассматриваться как более предсказуемая, а петля Бойда — как больше основанная на реакции.

Как последовательная разработка, так и Agile рассматривают как долгие сроки, так и короткие, но они смотрят на вопрос с разных сторон. При последовательной разработке составляется план на долгий срок, а краткосрочные работы выполняются уже внутри этого плана. В разработке по Agile делается упор на выполнение краткосрочных задач, то есть долгосрочные планы строят для того, чтобы предвидеть контекст для краткосрочного планирования.

Последовательная разработка рассматривает неопределенность как риск, как что-то, что нужно уменьшить или устранить. Петля Бойда рассматривает неопределенность как возможность, как что-то, что можно использовать для получения преимущества над соперником.

Различие между подходом последовательной разработки и Agile можно охарактеризовать как применение планирования, прогнозирования и контроля, с одной стороны, и наблюдение, оперативность и улучшение — с другой.

Ключевой принцип: изучайте и приспосабливайтесь

Я нахожу, что фраза «изучайте и приспосабливайтесь» — это полезное обобщение петли Бойда и также подходящей, эффективной цели в разработке по Agile. Команды, организованные по Agile, стараются избегать идеализма в своих методах и вместо этого подстраивают методы на основании эмпирических наблюдений за тем, что доказало свою эффективность. Команды, работающие по Agile, должны систематически все изучать и ко всему приспосабливаться, будь то планы, проектирование, качество кода, тесты, процессы, взаимодействие в команде — требуется уделять внимание каждому фактору, который может повлиять на

эффективность работы команды. Это не лицензия на приспосабливание без проверки. Изменения должны основываться на опыте.

Раздел «Рекомендации» в конце каждой главы подчеркивает значение этого принципа.

Рекомендации

Изучайте

- Пересмотрите ваши текущие проекты. Какие их составляющие можно отнести к «запутанным», а какие — к «сложным»?
- Пересмотрите недавно законченный проект. Как команды относились к проекту: как к «сложному» или как к «запутанному»? Оказывалось ли так, что какие-то вызовы, связанные с проектом, могли возникнуть из того, что «запутанный» проект могли классифицировать по ошибке как «сложный» (или наоборот)?
- В какой степени в ваших проектах применялись изучение и приспособление? Когда и где еще вы могли применить изучение и приспособление?
- С точки зрения петли Бойда понаблюдайте за тем, кто ваш соперник (определенный конкурент, доля рынка, требуемый уровень получения прибыли, бюрократия и так далее).
- Понаблюдайте за 3–5 областями неопределенности, которые вы могли бы подчинить своим интересам, чтобы получить преимущество над соперником.

Приспосабливайтесь

- Создайте список проектов, к которым в вашей организации должны относиться как к «запутанным», а не как к «сложным». Проведите работу с вашими проектными командами, для того чтобы приучить их относиться к ним соответственно.
- Ориентируйтесь с помощью областей неопределенности теми способами, которые обеспечивают преимущества перед соперником.
- Принимайте решение, как воспользоваться преимуществами ваших наблюдений, а затем действовать.

Дополнительные ресурсы

[Дэвид Дж. Сноуден и Мэри Э. Бун, 2007]. A Leadet’s Framework for Decision Making. Harvard Business Review. Ноябрь 2007 г. Это хорошо написанное введение в Cynefin, которое охватывает гораздо больше информации, чем дал я.

[Барри В. Боэм, 1988]. A Spiral Model of Software Development and Enhancement. IEEE Computer, май 1988 г. С точки зрения Cynefin эта работа предлагает подход к проектам, при котором каждый из них изначально рассматривается как «запутанный». Нужно проводить разведку, до тех пор пока полный объем вызовов, связанных с проектом, не будет достаточно понят, для того чтобы отнести проект к «сложным». На этом этапе проект выполняется с использованием подхода последовательной разработки.

[Стив Адольф, 2006]. What Lessons Can the Agile Community Learn from a Maverick Fighter Pilot? Заседание конференции Agile 2006. Здесь представлено краткое объяснение петли Бойда в особом контексте Agile.

[Джон Р. Бойд, 2007]. Patterns of Conflict. Январь 2007 г. Эта работа — воссоздание инструктажа полковника Джона Бойда.

[Роберт Корам, 2002]. Boyd: The Fighter Pilot Who Changed the Art of War. Здесь представлена подробная биография полковника Джона Бойда.

[Чет Ричардс, 2004]. Certain to Win: The Strategy of John Boyd, Applied to Business. Книга представляет собой понятное описание происхождения петли Бойда и ее применения для принятия управленческих решений.

ЧАСТЬ II. Еще более эффективные команды

В этой части рассказано о проблемах, касающихся людей и команд. Описана наиболее распространенная структура команд в Agile и Скраме. Затем обсуждаются команды Agile в целом: культура команд Agile, географически распределенные команды, а также личностные и коммуникативные навыки, которые поддерживают эффективность работы Agile.

Если вам больше интересны конкретные рабочие методы, чем динамика в команде, переходите к части III «Еще более эффективная работа». Если вам больше интересны проблемы менеджеров высшего звена, переходите к части IV «Еще более эффективная организация».

ГЛАВА 4. Начало эффективного Agile: Скрам

Основной проблемой в отрасли разработки ПО за 35 лет моей карьеры и, вероятно, дольше было избегание модели «написания кода, проверки и исправления ошибок» — то есть написание кода без продумывания или планирования с последующей отладкой, до тех пор пока он не станет работать. Такой неэффективный режим приводит к тому, что больше половины затраченных усилий у команд уходит на исправление недочетов, созданных ранее ими же [Макконнел, 2004].

В 1980–1990-х разработчики сказали бы, что занимаются структурным программированием, но многие действительно работали по модели «написания кода и исправления ошибок», тем самым упуская преимущества структурного программирования. В 1990–2000-х разработчики сказали бы, что занимаются объектно-ориентированным программированием, но многие из них все еще занимались тем, что писали код и исправляли ошибки, страдая от последствий. В 2000–2010-х разработчики и команды продолжают говорить, что занимаются разработкой по Agile, и хотя прошли уже десятилетия нарабатывания опыта и последствия известны, многие все еще продолжают заниматься

написанием кода и исправлением ошибок. Чем больше что-то меняется, тем больше остается таким же!

Вызов, идущий от Agile, заключается в том, что Agile открыто ориентирован на короткие сроки и сосредоточен на коде, поэтому становится трудно сказать, на самом ли деле команды работают по методам Agile или просто занимаются написанием кода и исправлением ошибок. Стена, плотно заклеенная стикерами, не обязательно означает, что команда в своей работе применяет организованный и эффективный подход. Когда подходы последовательной разработки скатываются в бюрократию, подходы Agile скатываются в анархию.

Одна из миссии этой книги — не допускать «постановочный» Agile, то есть случаи, когда команды, прикрываясь разработкой по Agile, только и занимаются тем, что пишут код и исправляют ошибки. Для начала хорош Скрам.

Ключевой принцип: начните со Скрама

Если вы все еще не внедрили Agile или если его реализация не так эффективна, как хотелось бы, я рекомендую отталкиваться от самого начала. В Agile это Скрам. Скрам наиболее директивный из распространенных Agile-методов. В его обширную экосистему входят множество книг, предложений по подготовке и инструментов. И есть доказательства, что он работает. Комплексный анализ исследований, проведенный Дэвидом Ф. Рико, показал, что в среднем рентабельность инвестиций во внедрение Скрама составляла 580% [Рико, 2009]. В докладе 2017–2018 годов State of Scrum сообщается, что в 84% случаев внедрения Agile применялись методы Скрам [Scrum Alliance, 2017].

Что такое Скрам?

Скрам — это легковесный подход к управлению рабочим процессом в командах, предусматривающий структуру и дисциплину. Скрам особо не предписывает никаких технических методов. Он позволяет определить, как будет проходить рабочий процесс в команде, и уже затем предписывает определенные роли и методы координирования работы, которые будет применять команда.

Основы Скрама

Если вы уже знакомы с основами, то можете пропустить этот раздел и перейти к разделу «Распространенные ошибки при внедрении Скрама», а если и эта тема вам уже знакома, то к разделу «Факторы успешного внедрения Скрам». Считается, что наиболее канонично Скрам описан в работе «Руководство по Скраму» [Швабер, 2017]. В основном опыт применения Скрама в моей компании совпадает с описанием, приведенным

в этом руководстве. Следующее описание соответствует руководству версии ноября 2017 года, если не оговорено иное.

В целом, Скрам известен тем, что уделяет внимание проведению мероприятий (встречи, митинги, церемонии), распределению ролей и наличию артефактов, связывая все это набором правил.

Работа по Скраму начинается с определения бэклога продукта владельцем продукта (Product Owner) — сотрудником, ответственным за требования. Бэклог продукта — это набор требований, в том числе выполняемых в настоящее время, функций, историй, улучшений и исправлений, который команда, работающая по Скраму, вероятно, может обеспечить. Вместо предложения полного списка всех возможных требований бэклог продукта фокусирует внимание на тех требованиях, которые наиболее важны, срочны и наиболее окупаемы (обеспечивают самый высокий возврат инвестиций).

Команды, работающие по Скраму, организуют работу в «спринты», которые представляют собой итерации длиной от 1 до 4 недель. Лучше всего подходят спринты длительностью в 1–3 недели. Практика показала, что риски увеличиваются с длительностью спринтов, а возможности улучшения ограничиваются. Чаще всего команды работают по двухнедельным спринтам.

Терминология, относящаяся к каденциям, то есть к некоему регулярному пульсу работы, может немного запутывать.

Термин «спринт» означает итерацию, которая условно представляет собой каденцию продолжительностью 1–3 недели.

Термин «развертывание» означает доставку продукта до потребителя, которая может происходить в сети за часы, а может занимать год и больше при встраивании ПО в оборудование. Разработку можно организовать по каденциям в 1–3 недели вне зависимости от того, сколько времени занимает доставка — часы, месяцы или годы.

Значение термина «релиз» разнится в зависимости от контекста, но чаще всего означает больший объем работы, чем выполняемый за спринт, то есть означает больший промежуток времени или больший набор связанного функционала.

На рис. 4.1 представлен порядок работ в проекте, выполняемом по Скраму.

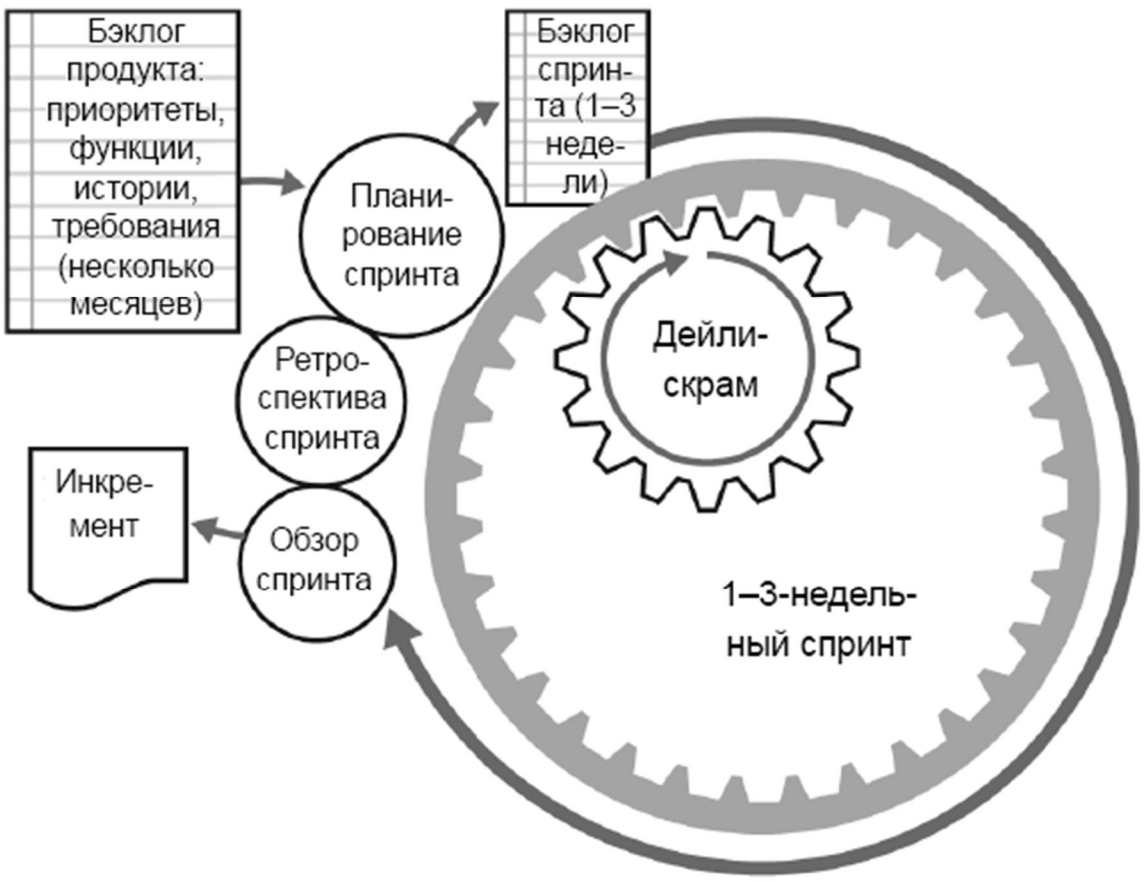


Рис. 4.1. Краткая схема работы по Скраму

Каждый спринт начинается со встречи по планированию спринта, во время которой команда, работающая по Скраму, пересматривает бэклог продукта, определяет объем работ для бэклога продукта в текущем спринте, поручает своим членам выполнить к концу спринта определенные задачи из бэклога продукта, а также составляет прочие планы, необходимые для проведения спринта.

Команда также определяет цель спринта, которая отражает то, на что направлено основное внимание. Если во время спринта происходят непредвиденные события, то цель представляет принципиальную основу для пересмотра деталей работы.

Над планированием спринта работает вся команда. Это эффективно, потому что команда кросс-функциональна и имеет все компетенции для принятия решений.

Встреча по планированию спринта не проходит без подготовки. Команда уточняет требования и обсуждает проектирование в достаточных подробностях до начала встречи, благодаря чему она проходит эффективно.

Функционал, представляемый командой в конце каждого спринта, называется инкрементом продукта. Обычно слово «инкремент» относится только к функционалу, добавленному во время каждого спринта в отдельности. Однако в Скраме инкремент относится ко всему функционалу, реализованному на сегодняшний день, в совокупности.

Во время самого спринта его бэклог считается неприкосновенным. Во время спринта проясняются требования, однако никто не имеет права добавлять, убирать или изменять требования, которые могут нарушить цель спринта, кроме случаев, когда владелец продукта согласен отменить спринт и начать цикл заново. На практике обычно отменяется несколько

спринтов. При смене приоритетов цели спринта и особенности иногда изменяются по взаимному согласию.

Во время спринта команда собирается на дейли-скрам (также известен как ежедневный стендап), он проводится каждый день, за исключением первого и последнего дня спринта. На 15-минутной встрече, сосредоточенной на отслеживании продвижения относительно цели спринта, обычно отвечают всего на три вопроса:

- Что было сделано вчера?
- Что будет сделано сегодня?
- Какие сложности в достижении результатов имеются?

Любое обсуждение, не касающееся трех перечисленных вопросов, как правило, откладывается на время после стендап-планерки, хотя некоторые команды предпочитают подход, более ориентированный на обсуждение.

Актуальное «Руководство по Скраму» гласит, что эти три вопроса на сегодня устарели, однако я считаю, что они придают встрече структуру, которая важна для плодотворного ее проведения.

Команда, применяющая Скрам, так и живет: стендап, работа, стендап, работа, «сполоснуть и повторить» — каждый спринт одно и то же.

Команды будут часто сверяться с диаграммой сгорания задач, чтобы отслеживать прогресс во время каждого спринта. Пример диаграммы — на рис. 4.2.

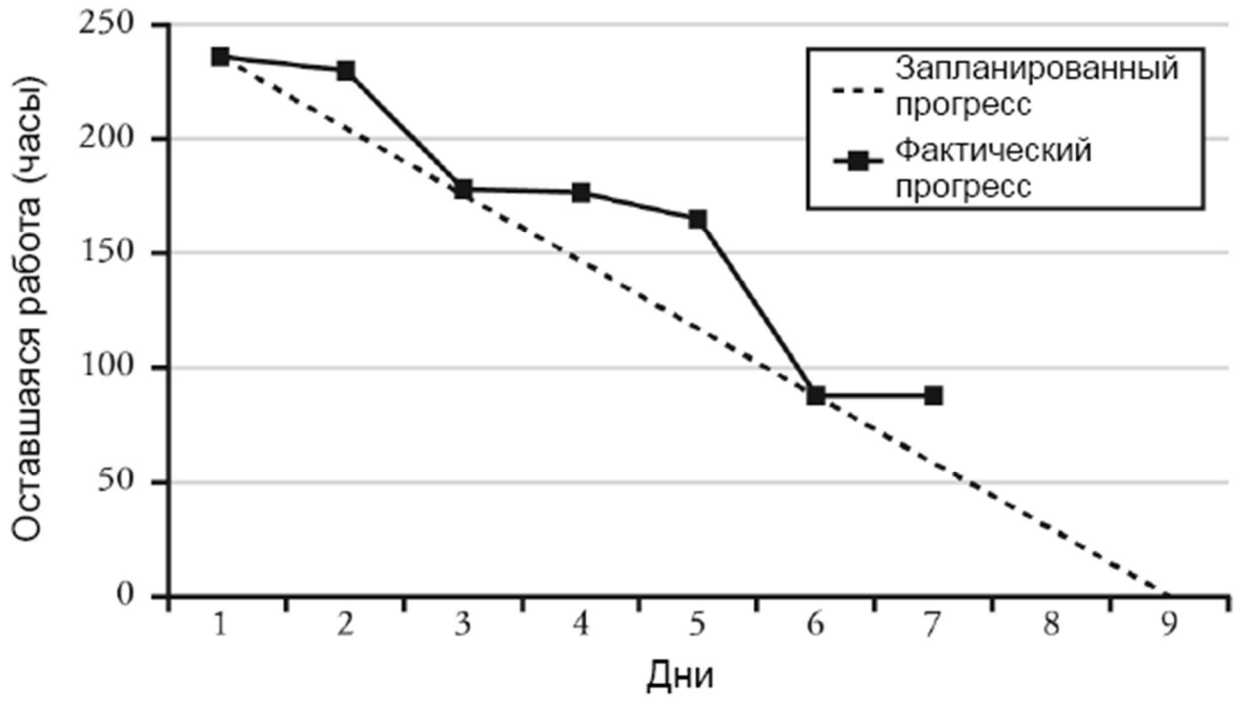


Рис. 4.2. Пример диаграммы сгорания задач, на которой показано соотношение запланированного и оставшегося объема работ в часах. Диаграмма сгорания задач, как правило, измеряет часы, а не истории

Диаграммы сгорания задач основаны на оценочных выводах и показывают количество часов, оставшееся на выполнение незавершенных задач, а не время, затраченное на выполненные задачи. Если планировалось, что задача займет 8 часов, а в действительности она занимает 15 часов, объем оставшейся работы на диаграмме сократится лишь на те 8 часов, которые были по плану. (По сути, это то же самое, что

управление освоенным объемом.) Если ожидания команды по выполнению плана на спринт завышены, то оставшиеся часы на диаграмме будут сгорать медленнее, чем ожидалось.

Некоторые команды предпочитают отслеживать продвижение работы во время спринтов не по часам, а по единицам сложности пользовательских историй. (Единицами сложности измеряют размер и сложность истории.) Цель диаграммы сгорания задач — ежедневное отслеживание продвижения хода работ. Если команда обычно выполняет минимум по одной истории в день, то по диаграмме можно будет отслеживать прогресс каждый день, а значит, можно отслеживать ход работ по историям. Если команда обычно выполняет одну историю лишь в два-три дня или выполняет большую часть историй ближе к концу спринта, то лучше измерять работу в часах, так как в этом случае по историям не получится отслеживать продвижение хода работ ежедневно.

Когда организации важна предсказуемость в течение долгого срока, мы рекомендуем вести еще диаграмму сгорания задач до релиза, чтобы отслеживать общий прогресс до текущего выпуска. Диаграмма сгорания задач до релиза условно показывает общее количество единиц сложности, в которое оцениваются истории, запланированные к релизу, скорость прогресса на данный момент и прогноз даты релиза.

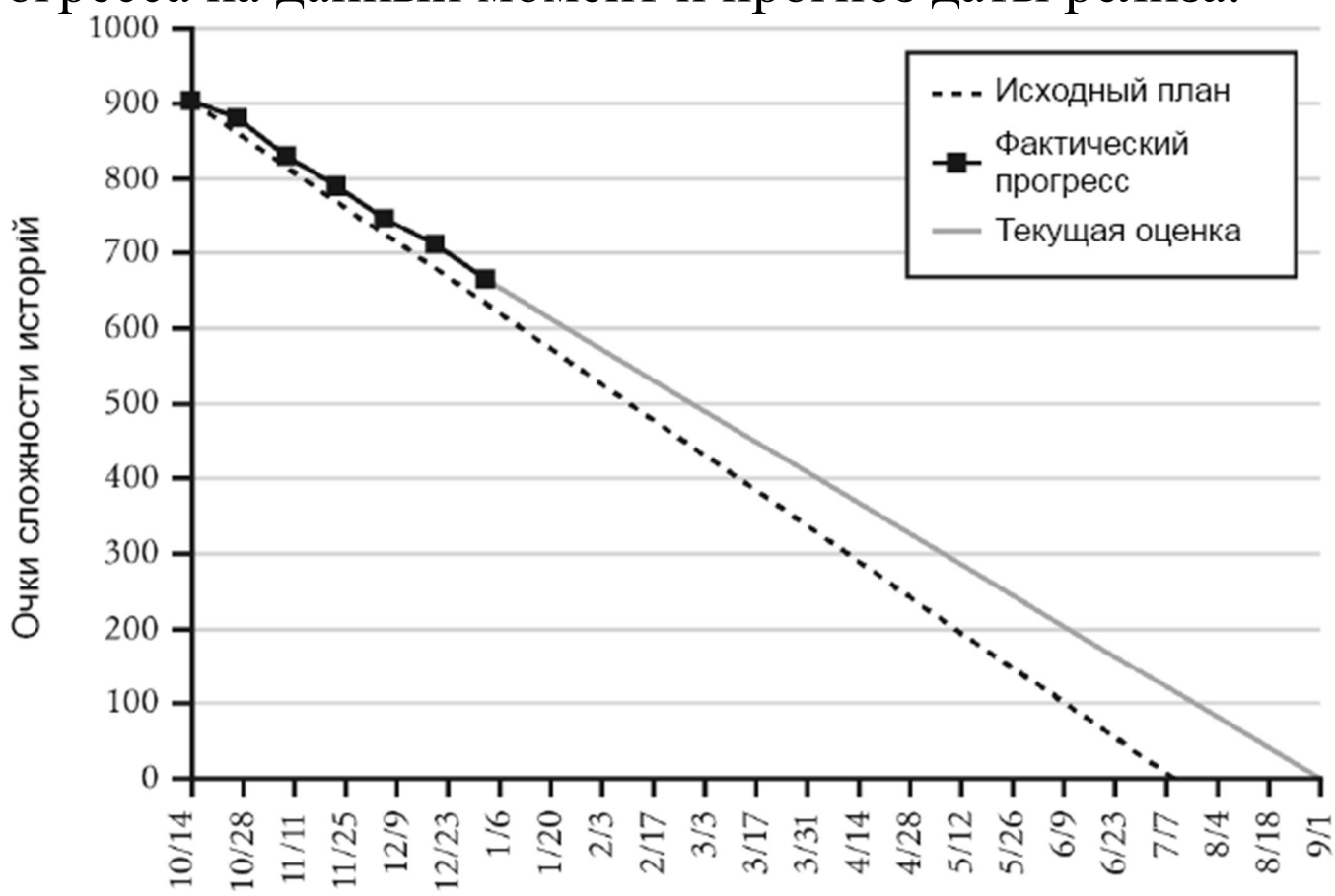


Рис. 4.3. Пример диаграммы сгорания задач до условного релиза

Возможны более информативные и подробные диаграммы. Они могут быть представлены в виде как диаграммы сгорания (нисходящий график), так и диаграммы выполнения задач (восходящий график). По ним можно отследить историю роста и снижения функциональности релиза, разброс ориентировочных сроков выполнения проекта и так далее. На рис. 4.4 приведен пример более сложной диаграммы выполнения задач.

В главе 20 «Еще более эффективная предсказуемость» содержится подробная информация о том, как поддерживать предсказуемость проектов, выполняемых по Agile.

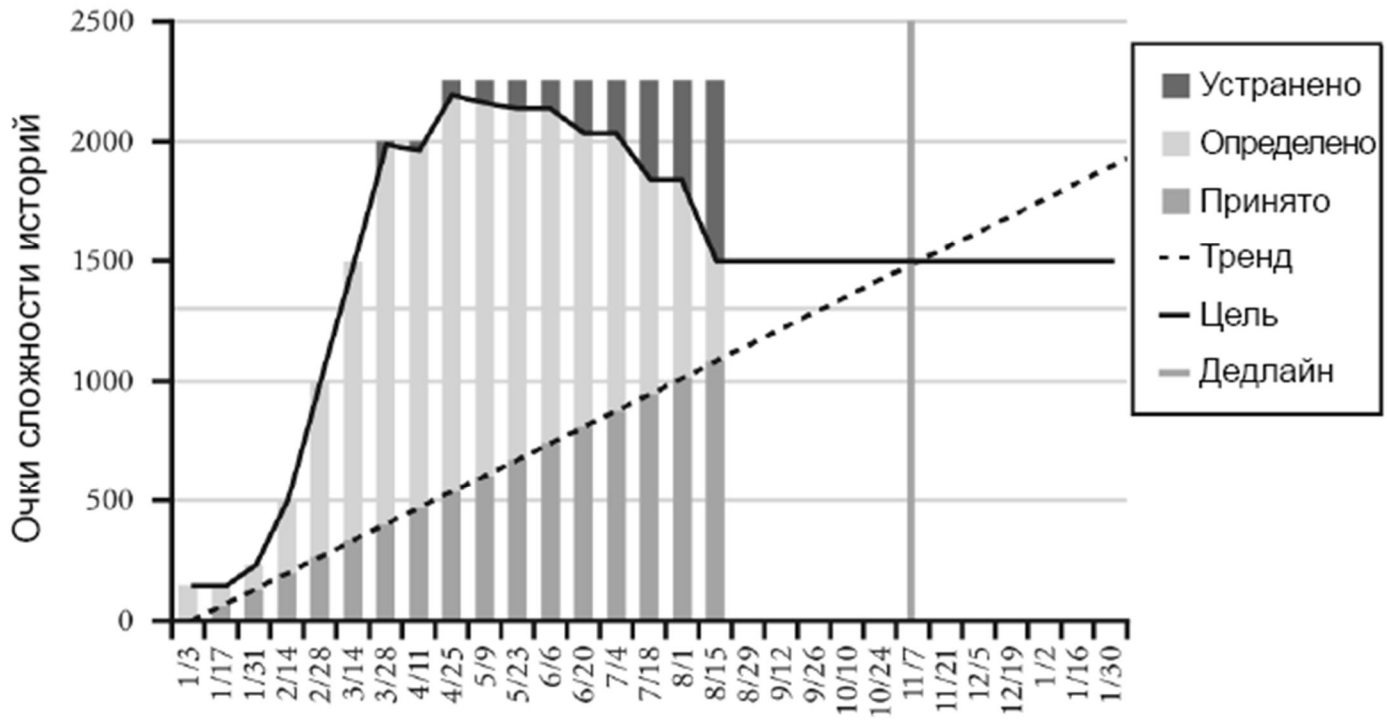


Рис. 4.4. Пример более сложной диаграммы выполнения задач до релиза

На протяжении спринта команда сохраняет высокий уровень качества работы. К концу спринта качество работы должно быть на уровне релиза и должно соответствовать так называемым «критериям готовности», на которые равняется команда (расскажу чуть позже). Команде не обязательно выпускать релиз в конце каждого спринта, но качество должно позволять в итоге выпустить то, что было реализовано в каждом спринте, без внесения дальнейших изменений.

В конце спринта команда, работающая по Скраму, демонстрирует реальные плоды своей работы на встрече, которая называется «обзор спринта», или «демо». Команда приглашает заинтересованные стороны (стейкхолдеров), чтобы обменяться мнениями и обеспечить обратную связь. Владелец продукта принимает или отклоняет работы, основываясь на заранее согласованных критериях приемки и обратной связи от стейкхолдеров, хотя эти решения нужно хорошо обдумать еще до обзора спринта. Команда использует обратную связь, полученную во время обзора спринта, чтобы улучшить продукт, а также процесс и методы его разработки.

Заключительный этап каждого спринта — «ретроспектива», на которой происходит обзор успехов и неудач текущего спринта. Здесь команда может использовать подход «Изучайте и приспосабливайтесь» для улучшения процесса разработки ПО. Команда проводит обзор всех изменений, внесенных ранее, и решает, стоит ли развивать то или иное изменение или лучше его откатить. Команда также оговаривает, какие новые изменения в процессе войдут в следующий спринт.

Роли в Скраме

В Скраме для обеспечения рабочего процесса есть три роли.

Владелец продукта представляет собой лицо, которое обеспечивает взаимодействие между командой с одной стороны и руководством, клиентами и заинтересованными сторонами, — с другой. Он несет основную ответственность за определение бэклога продукта и постановку приоритетов внутри этого бэклога, при этом он несет общую ответственность за определение состава и объема работ по продукту таким образом, чтобы команда работала с наибольшей отдачей. Владелец продукта регулярно уточняет бэклог продукта, сверяясь с командой, для того чтобы бэклог содержал уточненные (полностью определенные) задачи объемом примерно на два спринта, не считая объема задач в бэклоге текущего спринта.

Скрам-мастер отвечает за правильное выполнение методов Скрама. Он помогает команде и крупной организации усвоить теорию Скрама, практические методы и подход в целом. Он руководит процессом, при необходимости настаивает на его соблюдении, устраняет препятствия и играет роль коуча, оказывая поддержку остальным членам команды. Скрам-мастер может вносить вклад в техническую работу команды помимо своей основной роли, если ему на это хватает времени.

Команда разработчиков состоит из отдельных участников, каждый из которых кросс-функционален и работает непосредственно над задачами из бэклога.

Скрам-команда обычно состоит из 3–9 участников команды разработчиков, скрам-мастера и владельца продукта (рис. 4.5).

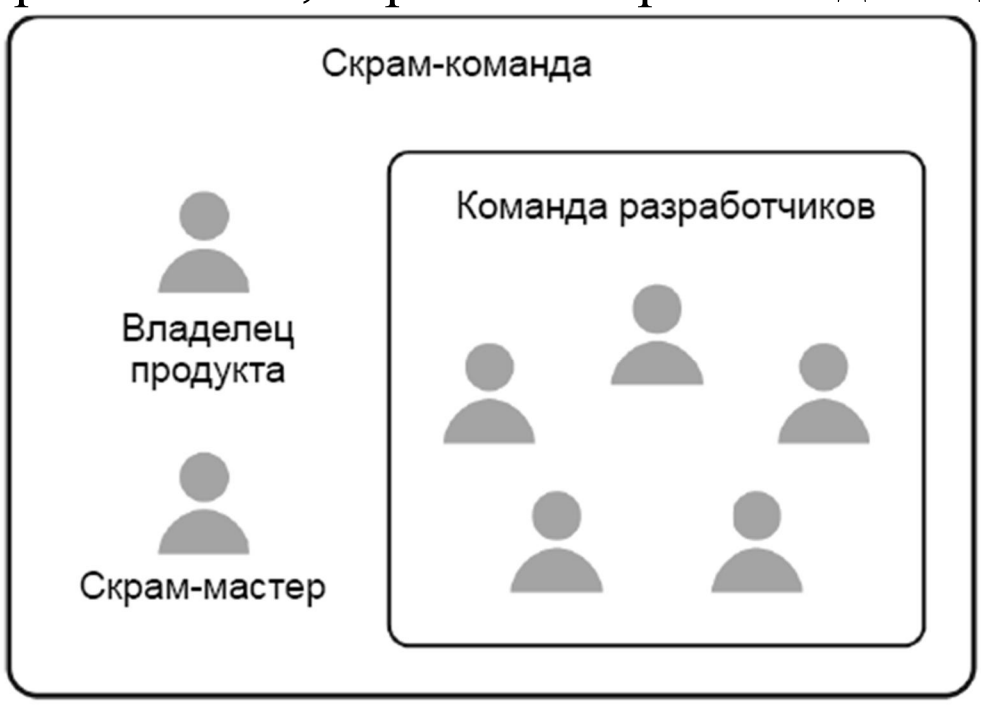


Рис. 4.5. Организация команды, работающей по Скраму. Иногда скрам-мастер является членом команды разработчиков, но не всегда

Обратите внимание: роли в Скраме — это именно роли, а не должности. Как сказал мне один топ-менеджер: «Наши должности не обусловлены ролями в Скраме. Нам не нужно, чтобы методы управления кадрами у нас зависели от технических методик».

Распространенные ошибки при внедрении Скрама

Я и сотрудники моей компании видели больше неэффективных случаев работы по Скраму, чем эффективных. Наименее эффективно Скрам применяли там, где были какие-то «но», что означает игнорирование его ключевых методов. Приведем несколько примеров: «Мы работаем по Скраму, *но* не проводим ежедневных стендапов», «Мы работаем по Скраму, *но* не проводим ретроспектив» или «Мы работаем по Скраму, *но* у нас нет владельца продукта». В неэффективных реализациях Скрама обычно отсутствует минимум один из его существенных атрибутов. А вот мой любимый пример: «Мы попробовали Скрам, но поняли, что большая часть практических методов не подходит для нашей организации. Мы работаем по Скраму, но главный практический метод, который мы применяем, — это ежедневные стендапы, мы проводим их каждую пятницу».

В отличие от разросшегося общего понятия *Agile*, Скрам — это минимальный набор методов для управления рабочим процессом. Поскольку Скрам уже представляет собой некоторый минимум методов, нельзя убрать из него какую-то часть и в итоге достигнуть успехов. Совершенство достигнуто не тогда, когда нечего добавить, а когда нечего убрать.
Антуан де Сент-Экзюпери
Если ваша организация внедрила Скрам и вы не видите значительных преимуществ, первый вопрос, который стоит задать: «А вы действительно внедрили Скрам или только некоторые его части?»

При продвинутом внедрении Скрама можно, в конце концов, убрать какие-то элементы, строго применяя к рабочему процессу методы изучения и приспособления. Но этим могут заниматься опытные пользователи, а не новички. Новичкам следует придерживаться всех правил.

В следующих разделах приводится описание наиболее распространенных сложностей, которые встречаются при внедрении Скрама.

Неэффективный владелец продукта

Десятилетия до появления Agile источником трудностей и неудач в проектах чаще всего называли плохо сформулированные требования. Так что неудивительно, что после появления Agile самой проблемной ролью в проектах, выполняемых по Скраму, является роль ответственного за требования.

Проблемы, касающиеся владельца продукта, бывают нескольких видов:

- Отсутствует владелец продукта — ожидается, что эту роль возьмет на себя кто-то из скрам-команды.

- Работы владельца продукта недостаточно — команде остро не хватает требований. Владелец продукта может поддерживать 1–2 команды, редко больше.
- Владелец продукта плохо понимает представителей бизнеса, а это выливается в низкокачественные требования или требования с неправильной расстановкой приоритетов, которые затем передаются всей команде.
- Владелец продукта не понимает, как обозначать требования к ПО, — это еще одна дорога к низкокачественным требованиям, которые получает команда.
- Владелец продукта не понимает технических трудностей, с которыми сталкивается команда разработчиков, поэтому не может эффективно расставить приоритеты для технических задач или навязывает подход в стиле «просто бери и делай», что приводит к накоплению технического долга.
- Владелец продукта работает отдельно от остальной команды, применяющей Скрам, — остальные члены команды не могут вовремя получить ответы на вопросы к требованиям.
- Владелец продукта не имеет права принимать решения по продукту.
- Цели и планы владельца продукта отличаются от целей бизнеса — владелец продукта дает команде такое направление, которое впоследствии отвергается бизнесом.
- Владелец продукта не представляет интересы обычных пользователей, к примеру владелец продукта — продвинутый пользователь и слишком вдается в мелочи.
- Владелец продукта не соблюдает правила, предписанные Скрам, — настаивает на изменении требований в середине спринта или ведет проект с прочими нарушениями методов Скрам.

Многие из этих проблем возникают из-за того, что бизнес не относится к роли владельца продукта так же серьезно, как относится к разработчикам или скрам-мастеру. Бизнесу следует относиться к роли владельца продукта как к самой серьезной в скрам-команде и уделять должное внимание наполнению этой роли. При соответствующей подготовке хороший владелец продукта может выйти из бывшего бизнес-аналитика, сотрудника службы поддержки клиентов или тестировщика. Ключевые характеристики владельца продукта, обладающего высокой производительностью, обсуждаются в главе 14 «Еще более эффективное определение приоритетов требований».

Недостаточное уточнение бэклога продукта

Бэклог продукта нужен, чтобы назначать задания команде разработчиков. Владелец продукта ответствен за бэклог продукта, а уточнять требования нужно непрерывно, чтобы у команды не было простоев в работе.

Уточнение бэклога продукта (также известное как «груминг бэклога») включает в себя настолько подробное изложение историй, чтобы было возможно их реализовать; разбиение слишком крупных историй на более мелкие, чтобы уложиться в спринт; добавление новых историй; обновление относительных приоритетов различных задач в бэклоге; оценку или переоценку историй и тому подобное. В общем, уточнение

бэклога состоит из определения подробностей, которые понадобятся команде, для того чтобы начать выполнять объем задач в следующем спринте. В этом могут помочь «критерии готовности задачи», описанные в главе 13 «Еще более эффективное создание требований».

Недостаточное уточнение бэклога может принести скрам-команде колоссальные проблемы. Хорошо уточненный бэклог продукта — это настолько важный вопрос для проектов, выполняемых по Agile, что в главах 13 и 14 проводится гораздо более подробный разбор.

Условно уточнение бэклога — работа для всей команды. Но поскольку за бэклог продукта отвечает владелец, а он выбран неправильно, то, как правило, уточнение бэклога продукта в проекте происходит тоже неправильно.

Слишком большие истории

Чтобы к концу каждого спринта работа могла выйти в релиз, необходимо выполнять истории в рамках одного спринта. В этой области нет никаких строгих правил, но есть два полезных совета:

- Команде нужно разделить истории так, чтобы ни одна история не занимала больше половины команды на половину спринта — большинство историй должны быть меньше.
- Команде нужно стремиться завершить 6–12 историй за каждый спринт (при условии, что команды рекомендованного размера).

Общая цель команды заключается в том, чтобы выполнять истории на протяжении всего спринта, а не только в последние пару дней.

Дейли-скрам проводится не каждый день

Ежедневные митинги приедаются, поэтому некоторые команды приходят к тому, чтобы проводить их три раза в неделю, а иногда и вовсе один. Но важно проводить их каждый день, чтобы дать членам команды возможность координировать работу, просить о помощи и отчитываться друг перед другом.

Чаще всего мы слышим, что дейли-скрам проводится реже потому, что «митинги занимают слишком много времени». В этом и соль проблемы! Встречи должны укладываться в 15 минут. Если сосредоточиться на трех ключевых вопросах, то в этот отрезок времени уложиться можно. Решение проблемы чрезмерно долгих ежедневных митингов — не уменьшить частоту их проведения, а ограничить время встречи и сосредоточиться на трех вопросах. Более подробно о ежедневных митингах можно узнать далее в этой главе.

Слишком долгие спринты

Считается, что лучше всего, чтобы спринты занимали от одной до трех недель, при этом большинство команд предпочитает двухнедельные спринты. Когда спринт длится дольше трех недель, появляется слишком большое раздолье для ошибок в планировании, чрезмерно оптимистичных обещаний, проволочек и прочего.

Акцент на горизонтальных, а не вертикальных срезах

Понятие «вертикальный срез» означает функциональность всего технологического стека от начала до конца. Понятие «горизонтальный срез» означает полезную возможность, которая прямо не позволяет продемонстрировать функционал на требуемом для бизнеса уровне. Выполнение работ вертикальными срезами помогает обеспечить более тесную обратную связь и более скорую разработку полезного для бизнеса функционала. Применение горизонтальных и вертикальных срезов — важная тема, которая раскрыта в в главе 9 «Еще более эффективное выполнение проектов».

Разделение команды разработчиков и тестировщиков

Частый пережиток модели последовательной разработки — разделение команды разработчиков и тестировщиков. Такое деление отнимает у скрам-команд кросс-функциональные навыки и знания, которые нужны для эффективной работы.

Нечеткие критерии готовности

Одно из важнейших средств поддержки высокого качества — это строгие критерии готовности (Definition of Done, DoD). Они гарантируют, что когда член команды или команда сообщает о том, что элемент бэклога готов, команда или организация может быть по-настоящему уверена в том, что больше ничего делать с этим элементом не требуется.

Критерии готовности — это эффективные выходные критерии, которые определяют стандарт, которому работа должна соответствовать для выпуска в производство, следующий этап интеграции или этап тестирования. Более подробно об этом рассказано в главе 11 «Еще более эффективное качество».

Уровень качества в спринтах не достигает уровня релиза

Одним из последствий избыточного давления сроков является то, что команды и их участники выдвигают на первый план имитацию прогресса, а не реальное выполнение. Поскольку качество не так очевидно, как базовая функциональность, команды под давлением сроков уделяют основное

внимание не качеству, а количеству. Случается, что они реализуют функциональность, которая содержится в бэклоге спринта, но не проводят тестирование, не создают автотесты или не выполняют другие нужные действия, чтобы обеспечить уровень качества, соответствующий релизу. Таким образом, работа объявляется «готовой», хотя некоторые задачи выполнены не полностью.

По нашим наблюдениям, более успешные Agile-команды не дожидаются конца спринта для достижения качества релиза. Они доводят каждую историю до требуемого качества, прежде чем приступать к следующей.

Не проводятся ретроспективы

Когда команда чувствует перегрузку из-за объема работ, за который она ответственна, то зачастую пропускает ретроспективы. Это огромная ошибка! Порочный круг чрезмерной самоотверженности и выгорания не будет разорван, пока вы не станете учиться на ошибках планирования и ведения работ, которые и привели к нему.

Agile-методы зависят от цикла «изучайте и приспосабливайтесь», а Скрам дает возможность делать это регулярно.

Результаты ретроспектив не учитываются в последующем спринте

Рассмотрим последнюю из самых распространенных ошибок. Чаще всего бывает, что команда проводит ретроспективы, но не извлекает из них уроков для следующего спринта. Знания накапливаются и оставляются на потом, и вместо того, чтобы сосредоточиться на принятии корректировочных мер, ретроспективы превращаются в сплошное нытье.

Не надо так. Проблемы нужно решать. Большинство проблем, влияющих на способность команд доставлять продукт, можно решить силами самой команды. Поддерживайте команды в принятии корректирующих действий с помощью ретроспектив, и вы будете поражены, как скоро они станут работать лучше. Более подробно ретроспективы обсуждаются в главе 19 «Еще более эффективное совершенствование процесса».

Скрам и что-то еще

Для начала достаточно одного Скрама, ничего больше не нужно. Некоторые команды добавляют шум в процесс ненужными дополнительными практическими методами. В одной из компаний, с которой мы работали, нам сказали: «Первая команда, применявшая Скрам, работала отлично, но после этого мы не смогли найти другую команду, которая была бы готова применять парное программирование или могла бы разобраться, как при нашей устаревшей конъюнктуре выполнять

непрерывную интеграцию». Ни парное программирование, ни непрерывная интеграция не являются требованиями Скрама. После того как в этой организации осознали, что ее команды могут применять Скрам без парного программирования и непрерывной интеграции, им удалось расширить применение Скрама.

Неэффективный скрам-мастер

Человек, ответственный за избегание таких ошибок, — скрам-мастер. Проблемы, связанные с ним, схожи с некоторыми проблемами, связанными с владельцем продукта:

- Отсутствует скрам-мастер — ожидается, что команда будет работать по Скраму без назначения определенного лица на роль мастера.
- Работы скрам-мастера недостаточно, он отвечает за поддержку слишком большого количества команд.
- Скрам-мастер совмещает свою деятельность с разработкой и ставит разработку выше по приоритету, чем работу скрам-мастера.
- Скрам-мастер недостаточно компетентен в своей области, чтобы наставлять команду и стейкхолдеров.

Может казаться очевидным, что роль скрам-мастера критически важна для эффективного внедрения Скрама, но нам часто встречались организации, где этой роли не было. Многих проблем, описанных здесь, можно избежать благодаря эффективной работе скрам-мастера.

Что общего в ошибках внедрения Скрама?

Все ошибки, которые я только что описал, являются разновидностями различных «но» при работе по Скраму. Первым делом команде или компании, внедряющей Agile, требуется убедиться, что Скрам применяется с высоким качеством.

У большинства этих ошибок есть другое общее свойство: отсутствие постоянного применения практических методов, требующих высокой дисциплины. Метод считается требующим высокой дисциплины, и люди имеют тенденцию отклоняться от него в том случае, когда нет поддержки структуры или социума, чтобы обеспечить выполнение такого метода.

Скрам-мастер отвечает за применение командой практических методов Скрама, требующих высокой дисциплины (и не только этих методов). Митинги в Скраме, такие как планирование спринта, дейли-скрам, обзор спринта и ретроспектива спринта, обеспечивают как социальную, так и структурную поддержку методов, где требуется высокий уровень дисциплины.

Факторы успешного внедрения Скрама

Каждую из ошибок можно обратить на пользу и сделать одним из факторов успешного внедрения. Перечислим некоторые примеры:

- Эффективный владелец продукта.
- Уточнение бэклога.
- Соблюдение небольшого размера историй.
- Ежедневное проведение митингов.
- Ограничение длительности спринтов до 1–3 недель.
- Организация работы по вертикальным срезам.
- Интеграция специалистов по тестированию и по контролю качества в команду разработчиков.
- Формулирование четких критериев готовности.
- Качество, приемлемое для релиза, в каждом спринте.
- Проведение ретроспектив в каждом спринте.
- Своевременное применение знаний и навыков, полученных из ретроспектив.
- Эффективный скрам-мастер.

Подробнее об этом — в следующих главах.

Успешный спринт

Успешным можно назвать спринт, соответствующий главной цели Скрама, которая заключается в доставке продукта наиболее высокого качества. На уровне спринта такая цель включает в себя следующее:

- Спринт обеспечивает пригодный и ценный инкремент продукта (совокупную функциональность), который полностью отвечает критериям готовности.
- Ценность инкремента спринта возрастает по сравнению с предыдущим спринтом.
- Скрам-команда улучшает процесс в сравнении с предыдущим спринтом.
- Команда узнает что-то новое о себе, бизнесе, своем продукте или своих клиентах.
- Команда мотивирована не хуже или лучше, чем в конце предыдущего спринта.

Распределение времени в типичном спринте

В этой главе мы обсудили полный спектр деятельности, которая происходит во время работы по Скраму. Легко прийти к выводу, что в Скраме не так много разработки, как кажется. В табл. 4.1 приведен типичный пример распределения сил разработчиков в скрам-команде для двухнедельного спринта.

Таблица 4.1. Пример распределения сил разработчиков во время спринта

Параметры планирования спринта	
Продолжительность спринта (рабочих дней)	10
Идеальное количество часов в сутки (время, потраченное на проект)	× 6
Общее идеальное количество часов на спринт для разработчика	= 60
Количество действий, по Скраму на спринт для разработчика	**часов**
Разработка, включая тестирование	48
Дейли-скрам (стендапы)	2
Уточнение бэклога продукта (5%)	3
Планирование спринта	4
Обзор спринта	2
Ретроспектива спринта	1
Итого	60

Понятие «идеальное количество часов» означает количество часов, затраченных на проект (которые остаются после времени, затраченного на корпоративные накладные расходы). Идеальное количество в 5–6 часов характерно для большой солидной фирмы. В небольших компаниях идеальное количество в среднем 6–7 часов, а в стартапах — и более того.

При идеальном количестве 60 часов в спринте около 20% времени уходит на планирование и улучшение процесса. Соответственно 80% остается на саму разработку.

Проблемы при переходе на Скрам

Командам нужно учиться решать практические проблемы внедрения: географическую разрозненность, устаревшие системы, поддержку продукта, трудности поиска ролей и так далее.

В начале внедрения команде может казаться, что работа замедлилась. В реальности команде чаще встречается работа, которую нужно выполнять как можно быстрее (раньше при последовательной разработке к концу проекта накапливались невыполненные задачи или такая работа просто проходила незаметно). По мере того как команда будет учиться работать пошагово, производя инкремент, ощущаемая скорость будет расти.

Система показателей Скрама

Чтобы оценить точность следования правилам Скрама при его внедрении, будет полезным создать систему показателей для проекта, по которой будут оцениваться наиболее важные факторы успешного внедрения. На рис. 4.6 изображен пример звездообразной диаграммы системы показателей проекта, который мы уже видели в главе 1.

Пояснения к диаграмме:

0 Не применяется

2 Применяется редко и неэффективно

4 Применяется эпизодически с переменной эффективностью

7 Применяется постоянно и эффективно

10 Оптимизация

Жирная линия показывает средние показатели при реализации практических методов, которые встречались моей компании при консультировании и обучении, выборка с 2010 года со смещением к наблюдениям за последние два года — примерно 1000 команд.

Пунктирная линия отражает показатели здоровой команды. Как я и писал ранее, по моим наблюдениям, средняя команда не очень хорошо соблюдает правила. Здоровая, эффективная команда при работе по Скраму наберет 7 пунктов и больше по всем факторам успешного внедрения.

Рис. 4.6. Инструмент диагностики, показывающий производительность команды при работе по Скраму в соответствии с ключевыми факторами его успешного внедрения

Изучайте и приспосабливайтесь в Скраме: дейли-скрам

Со временем эффективная команда будет изучать контекст и приспосабливаться к имеющейся реализации Скрама. На начальном этапе внедрения нужно все делать по правилам, приспосабливаясь по ходу дела.

Наиболее часто команда пытается изменить что-нибудь в области дейли-скрама, вероятно, потому, что он проводится наиболее часто и поэтому предлагает возможность для размышления и совершенствования.

Нам встречались команды, которые разными способами меняли три ключевых вопроса. Вот несколько вариаций первого вопроса:

- Что было сделано вчера? (изначальный вопрос)
- Какие задачи были выполнены вчера?
- Что из того, что вы вчера выполнили, соответствует критериям готовности?
- Что вы сделали вчера, для того чтобы приблизиться к цели спринта?
- Что вы сделали вчера, для того чтобы выполнить план на спринт?

Команды оттачивают способ проведения дейли-скрама. Некоторые команды выводят три ключевых вопроса на экран, чтобы обсуждение не отклонялось от темы. В некоторых командах используется ораторский

жезл с этой же целью. Некоторые команды отклоняются от этих трех вопросов и предпочитают подход, больше ориентированный на дискуссию. Такие изменения можно считать полезными, пока команда отслеживает, ведет ли каждое из них к совершенствованию.

Прочие соображения

Отличительной чертой разработки по Agile стало распространение перечисленных практических методов. Каждый из таких методов был изобретен умными людьми — консультантами и практиками — не просто так. Каждый из этих методов хорошо сработал хотя бы один раз по меньшей мере в одной организации. У каждого из этих методов есть свои сторонники.

Эта книга сосредоточена на проверенных практических методах, которые широко применялись и работали во многих организациях. Далее в разделах «Прочие соображения» будут описаны некоторые практические методы, о которых вы, наверное, слышали, но которые, по опыту моей компании, сложно назвать проверенными и широко применимыми.

Экстремальное программирование (XP)

В начале существования Agile большое внимание уделялось экстремальному программированию (XP) (Бек, 2000; 2005), которое представляет собой набор определенных технических методов, процессов и дисциплин, олицетворявших изначальные принципы Agile. Первоначальное внимание на XP, если верить рекламе, было колоссальным, однако долгосрочное применение XP как целостного подхода к разработке ни у кого не получилось. Когда XP версии 1 требовало применения набора из 12 практических методов, даже в тех проектах, которые считались образцовыми, использовали примерно половину из этих методов [Греннинг, 2001; Шу, 2001; Пул, 2001].

Упор на полноценном использовании XP снизился с начала 2000-х годов. Сегодня XP вносит свой вклад как источник технических методов, которые считаются неотъемлемой частью современного Agile. К этим методам относится непрерывная интеграция, рефакторинг, разработка через тестирование и непрерывное тестирование.

Kanban

Kanban — это система планирования и управления задачами, основанная на прохождении разработки через несколько стадий. В Kanban делается акцент на перетягивании работы на последующие стадии, а не ее проталкивании с ранних стадий. Kanban дает возможность визуализировать работу, сократить количество выполняемых работ и достичь максимального потока работ через систему.

С точки зрения Cynefin Kanban хорошо подходит для «сложных» работ, где постановка приоритетов и пропускная способность команды имеют первостепенное значение, в то же время Скрам лучше подходит для «запутанных» работ, потому что в нем делается акцент на продвижении к главной цели малыми итерациями. Обе методологии могут послужить отличным фундаментом для улучшения процесса.

Kanban может оказаться более подходящим, чем Скрам, для небольших команд (1–4 человека) или для проведения работ, которые больше ориентированы на производство, чем на проектную деятельность.

Скрам-команды часто движутся в направлении интеграции Kanban в свою реализацию Скрама по мере лучшего овладения практическими методами Agile, а некоторые организации успешно применяли Kanban в качестве инструмента управления портфолио в масштабных проектах.

Некоторым командам удавалось успешно начинать внедрение Agile с Kanban. Но Скрам более структурирован, перспективнее, больше ориентирован на работу в команде, поэтому, как правило, он лучше подходит для того, чтобы с него начинать внедрение Agile.

Немного подробнее о Kanban мы поговорим в главе 19 «Еще более эффективное совершенствование процесса».

Рекомендации

Изучайте

- Опросите ваши команды о том, как они применяют Скрам. Предложите им оценить степень применения Скрама согласно системе показателей. Насколько эффективно применяется Скрам?
- Пересмотрите ошибки внедрения Скрама, приведенные в этой главе, с ключевыми игроками вашей команды и определите, в каких областях нужно провести улучшения.
- Проведите анализ команды. Посмотрите, кто подходит на роль скрам-мастера. Достаточно ли эффективно скрам-мастера помогают командам в применении практических методов, в том числе методов, требующих высокой дисциплины и связанных с ошибками внедрения Скрама?

Приспосабливайтесь

- Настаивайте на том, чтобы ваши команды применяли Скрам в точности по правилам, до тех пор пока вы не увидите у них количественно измеряемые показатели, которые могут послужить основой для применения чего-то нового (в главе 19 «Еще более эффективное совершенствование процесса» приводится информация, как измерить изменения, происходящие в рабочем процессе Agile).
- Если ваши скрам-мастера неэффективны, подумайте об их переподготовке или замене.

Дополнительные ресурсы

[Кен Швабер и Джефф Сазерленд, 2017]. Руководство по Скраму. Исчерпывающее руководство по Скраму: Правила игры. По мнению многих специалистов, это краткое руководство является исчерпывающим описанием практических методов Скрама.

[Кеннет Рубин, 2012]. Essential Scrum: A Practical Guide to the Most Popular Agile Process. Это издание является всеобъемлющим руководством по Скраму, которое рассматривает распространенные проблемы, связанные с внедрением Скрама.

[Митч Лейси, 2016]. The Scrum Field Guide: Agile Advice for Your First Year and Beyond, 2nd Ed. В этом руководстве по Скраму по косточкам разбираются проблемы, возникающие на практике при внедрении и применении Скрама.

[Майк Кон, 2010]. Succeeding with Agile: Software Development Using Scrum. Еще одна неплохая альтернатива работе Рубина 2012 г. и работе Лейси 2016 г.

[Джефф Сазерленд, 2014]. Скрам. Революционный метод управления проектами (М.: Манн, Иванов и Фербер, 2016). Эта книга ориентирована на бизнес и представляет читателю историю Скрама.

[Дженни Стюарт и др.]. Six Things Every Software Executive Should Know about Scrum. Серия Construx White Paper. Июль 2018 г. Здесь представлен краткий обзор Скрама, ориентированный на руководителей высшего звена.

[Дженни Стюарт и др.]. Staffing Scrum Roles. Серия Construx White Paper. Август 2017 г. В статье представлены распространенные проблемы при заполнении ролей в Скраме.

ГЛАВА 5. Еще более эффективная структура команды

Базовая единица производительности в методологии гибкой разработки Agile — не столько высокопроизводительные специалисты, сколько команды с высокой производительностью. Эта мысль ключевая, нам встречалось много организаций, которые подрывали у себя внедрение Agile с самого начала, не понимая, что нужно для успешной работы команды по Agile, и не оказывая своим командам должной поддержки.

В этой главе обсуждаются проблемы структуры команд, работающих по Agile, а в следующей главе рассказывается о культуре труда в команде, применяющей Agile.

Ключевой принцип: обеспечьте кросс-функциональность команд

В докладе *Accelerate: State of DevOps*2018 г. сообщается: «Существует двойная вероятность, что команды с высокой производительностью разрабатывают

и доставляют продукт, сотрудничая в единой, *кросс-функциональной команде* <…> по нашим наблюдениям, команды с низкой производительностью в два раза чаще разрабатывают и доставляют продукт, работая *отдельными изолированными командами*, чем команды с высокими результатами» (курсив мой) [DevOps Research and Assessment, 2018].

Эффективная команда, работающая по Agile, обладает функционалом, необходимым для автономной работы (такая команда в значительной степени самоуправляема). В контексте «Запутанные» по Cynefin бˆольшая часть работы команды состоит из цепочки *исследуйте.осознайте.реагируйте*. Если команде требуется обращаться к сторонним специалистам каждый раз, когда нужно исследовать или осознавать, она не сможет своевременно отреагировать. Команда должна быть в состоянии принимать большинство решений, связанных со своей работой, самостоятельно, в том числе решения об особенностях (требованиях) продукта, технических подробностях и тонкостях процесса. Основной массе программистов, пишущих окончательный код, также нужно создавать много кода для автотестов и разбирать подробные требования. Такая команда сможет быстро продвигаться в сложных условиях и при этом удовлетворять потребности бизнеса.

Самоуправляемая кросс-функциональная команда, как правило, требует следующих специалистов:

- Разработчики разных уровней приложения (фронтенд, бэкенд и т.д.) и с разной компетенцией (архитектура, пользовательский опыт, безопасность и т.д.).
- Тестировщики разных уровней приложения.
- Разработчики документации.
- Эксперты в области используемого процесса разработки (скрам-мастер).
- Специалисты узкого профиля.
- Бизнес-эксперты, которые помогут понять ожидания бизнеса, его мировоззрение и сделать продукт команды рентабельным (владелец продукта).

Трудно собрать команду, в которой будут специалисты со всем набором необходимых навыков, так чтобы не превысить рекомендованный состав в 5–9 человек. Одним и тем же людям нужно выполнять несколько ролей, и многим компаниям нужно помочь своим сотрудникам развить дополнительные навыки. В главе 8 «Еще более эффективные сотрудники и их взаимодействие» описаны практические методы, которые для этого необходимы.

Помимо навыков у высокопроизводительной кросс-функциональной команды должны быть возможность и полномочия для своевременного принятия необходимых решений.

Способность принимать решения

Способность принимать решения в большой степени зависит от того, как подобраны члены команды. Обладает ли команда полнотой компетенции для принятия эффективных решений? Достаточно ли команда компетентна в построении архитектуры, обеспечении качества, разработке продукта, во взаимодействии с клиентами и бизнесом? Или требуется обращаться к третьей стороне за экспертной помощью?

Команда, которой не хватает компетенции в какой-либо из этих областей, не может стать эффективной и кросс-функциональной. Она будет часто натыкаться на области, в которых ей не будет хватать компетенции для принятия решений. Чтобы получить рекомендации, команде впоследствии придется обращаться куда-то еще. Это приводит к многочисленным задержкам. Членам команды не всегда известно, к кому нужно обратиться, поэтому понадобится время и на то, чтобы найти нужного специалиста, а он не всегда может ответить сразу. Еще нужно время для того, чтобы объяснить контекст проблемы. Если сведения, полученные от такого специалиста, команде непонятны и нуждаются в разъяснениях, то подобная обратная связь может занять долгое время из-за задержек по тем же причинам. И команда, и сторонний специалист будут делать предположения, некоторые из них окажутся ошибочными, а такие ошибки потребуют еще времени на их поиски и исправления.

Иногда команде нужно обращаться за сторонней помощью, но если команда обладает всей необходимой компетенцией, то решать проблемы может самостоятельно за считаные минуты. Команду нужно собрать так, чтобы она могла решать большую часть вопросов сама.

В команде из 5–9 человек не может быть бесконечного количества специалистов. Стандартное решение: встраивание в команду внештатных специалистов на неполную ставку, например в области UX или архитектуры, на несколько спринтов.

Готовность укомплектовать Agile-команды специалистами, важными для принятия большинства решений на местном уровне, является ключевой проблемой при внедрении Agile.

Полномочия принимать решения в рамках своей компетенции

Полномочия принимать решения частично исходят от основных стейкхолдеров, все из которых должны быть представлены в команде, и частично от организации, которая наделяет команду соответствующими разрешениями. Чтобы команда работала эффективно, ей нужна возможность принимать *обязательные решения*, которые никто больше в организации не может отменить.

Нехватка необходимых полномочий порождает ряд проблем, которые приводят к снижению производительности:

- Команде придется слишком много времени затрачивать на проработку решений, которые в компании постоянно кем-то отменяются.
- Команде придется работать, слишком тщательно обдумывая каждый шаг, постоянно оглядываясь в ожидании того, что кто-то пересмотрит или отменит принятые ей решения.
- Команда будет периодически простаивать, из-за того что нужно добиваться утверждения решений от других членов организации.

Полномочия и возможности нужно рассматривать в совокупности. Предоставлять команде право принимать решения, если для этого не создается необходимых условий, неэффективно для организации. Если в команде действительно представлены интересы всех сторон, каждое решение будут рассматривать с точки зрения всех, кто относится к делу. Это не означает, что команда никогда не допустит ошибок. Это значит, что у команды будут прочные основания для принятия решений, а у остальной части компании будут прочные основания таким решениям доверять.

Нежелание организации делегировать команде полномочия на принятие необходимых решений — это еще один гвоздь в крышку гроба как для самой команды, так и для внедрения Agile.

Самоуправление в командах

По-настоящему самоуправляемую команду нельзя просто взять и создать, ее можно только взрастить. Не всегда с самого первого дня своей работы команда готова к самоуправлению.

Отчасти задача руководителя заключается в том, чтобы понять, когда команда достигает зрелости, и помочь команде обрести возможность принимать решения самостоятельно.

Важность ошибок

Как и любая другая самоуправляемая команда, команда, работающая по Agile, будет допускать ошибки. Будет здорово, если в организации присутствует эффективная культура обучения. С одной стороны, команда будет учиться на своих ошибках и работать лучше. С другой стороны, появляется мощная мотивация, если команда знает, что организация достаточно ей доверяет и у нее есть право на ошибку.

Организация тестировщиков

Организация тестировщиков на протяжении всей моей карьеры была движущейся мишенью. Однажды тестировщиков интегрировали в команды разработчиков, и те стали отчитываться перед менеджером по разработке.

Это породило проблему, так как менеджеры по разработке стали наседать на тестировщиков, говоря им: «Да откуда же вы берете столько багов? Может, хватит?» И получилось так, что вместо них баги стали находить сами клиенты.

Затем тестировщиков снова организовали в отдельные группы, обособив им отдельные зоны. Тестировщики больше не были подотчетны менеджеру по разработке. У них появилась другая структура отчетности, которая, как правило, не имела ничего общего с той структурой, которая бытовала у разработчиков, и доходила до уровня директора или вице-президента.

Из-за этой структуры появились новые проблемы, в том числе противостояние разработчиков и тестировщиков. Это противостояние обострялось тем, что тестировщиков считали цензорами, поскольку они прямо или косвенно отвечали за недопущение релизов низкого качества. Разделение ответственности за разработку и тестирование породило тенденцию к тому, что разработчики отказывались нести ответственность за тестирование своего же собственного кода.

Потом наступил следующий этап эволюции организации тестировщиков: они по-прежнему отчитывались отдельно, но их размещали в одном пространстве с разработчиками в целях укрепления сотрудничества. Разработчики передавали тестировщикам закрытые редакции сборок на тестирование, тестировщики писали тестовые сценарии и передавали их разработчикам. Разработчики пропускали свой код через тестовые сценарии и исправляли большинство ошибок до того, как официально регистрировали изменения в коде. Такой подход неплохо работал, позволяя свести к минимуму временной разрыв между появлением и обнаружением ошибок.

Ключевой принцип: интегрируйте тестировщиков в команды разработчиков

Сегодня на подход к организации тестирования влияют два фактора — возрастание роли Agile и развитие автоматизированного тестирования.

Agile поощряет разработчиков тестировать свою работу. Это важный шаг к уменьшению интервала между появлением и обнаружением недочетов. К сожалению, это привело к тому, что некоторые организации вовсе упразднили тестирование как отдельную специализацию. А это неправильно. Тестирование ПО — неимоверно глубокая область знаний. Не понимая фундаментальных концепций тестирования, большинство разработчиков зацикливаются на инструментах тестирования, не применяя даже фундаментальных практических методов, не говоря уже о более продвинутых.

Специалисты по тестированию до сих пор нужны, чтобы выполнять несколько задач:

- Нести основную ответственность за автоматизацию тестов.
- Создавать и сопровождать более сложные виды тестов, например стресс-тесты, тесты производительности, нагрузочные тесты и так далее.
- Применять более сложные практические методы тестирования, чем те, которые применяют разработчики, например покрытие области входных значений, анализ классов эквивалентности, покрытие граничных значений, покрытие диаграммы состояний, тестирование на основе рисков и так далее.
- Создавать тесты, которые упускают разработчики при тестировании своего собственного кода из-за замыленности взгляда.

Тестирование кода разработчиком является основополагающим в Agile, однако тестировщики все равно вносят свой вклад. В организациях, где эту должность упразднили, мы видим, как члены команды, которые раньше считались тестировщиками, главным образом концентрируются на интеграционных, нагрузочных и прочих междисциплинарных тестах. Также часто можно заметить, что они берут на себя большую долю автоматизации тестирования, чем прочие члены команды, занимающиеся самой разработкой.

В Agile есть практика «три товарища» (Three Amigos), которая подразумевает бизнес, разработку и как раз таки тестирование. В структурной схеме организации могут отсутствовать тестировщики, но на деле они никуда не исчезли. Это неявное признание пользы, которую приносят такие специалисты.

Как уже говорилось выше, эффективность разработки по Agile зависит от кросс-функциональности команд, включающих в себя и тестировщиков. Они должны работать плечом к плечу с разработчиками на протяжении всего процесса разработки и доставки ПО.

Организация поддержки продукта

Не могу вспомнить ни одной компании из тех, с которыми мы работали, которая была бы на 100% довольна тем, как в ней организована поддержка продукта.

Компании используют все или некоторые из следующих моделей:

- Поддержкой полностью занимаются специалисты, участвующие в создании продукта.
- Поддержкой полностью занимается специально выделенная команда.
- Поддержкой первого и второго уровней занимается отдельная команда, инженеры отвечают за поддержку третьего уровня.

Последний случай самый распространенный, и существует много разных подходов. Например, поддержку третьего уровня оказывает отдельная команда (которая состоит из специалистов более продвинутой технической компетенции, чем команда поддержки первого и второго уровней). Такая команда в первую очередь отвечает за поддержку продукта. Другой способ

обеспечения поддержки третьего уровня — сотрудниками, которые изначально занимались созданием системы, даже в том случае, когда они уже работают над другими продуктами.

В командах разработчиков, где рассматривают поддержку по назревшим проблемам в качестве вторичной обязанности (то есть разработчики ведут поддержку продуктов, над которыми работали ранее), поддержка может быть организована различными способами:

- Поддержку по назревшим проблемам оказывают все члены команды по мере поступления таких проблем по циклическому принципу.
- Поддержку по всем назревшим проблемам оказывает один из членов команды, который сменяется ежедневно или еженедельно.
- Поддержку по назревшим проблемам оказывают члены команды, которые наиболее квалифицированы в решении той или иной проблемы.

Со временем большинство компаний пробует некоторые из перечисленных моделей и приходит к выводу, что у всех способов есть свои недостатки. Цель состоит в том, чтобы из всех зол выбрать наименьшее, а не найти безупречное решение.

Поддержка продукта при работе по Скраму

В свете проблем поддержки продукта, характерных для Agile, задача состоит в том, чтобы обеспечивать поддержку без нарушения течения спринтов. Командам нужно предусмотреть и распланировать время, которое будет выделено на поддержку назревших проблем. Дадим несколько полезных советов:

Планируйте время на поддержку для каждого спринта. Если поддержка продукта занимает 20% рабочего времени команды, в плане на спринт допускается распределение только 80% времени на работу, относящуюся непосредственно к спринту.

Задайте политику на ту работу, которую можно делать вне рамок спринта. Проведите разграничение между обычной работой, которая может попасть в бэклог продукта, и проблемами, которые и достаточно срочные, и достаточно важные, чтобы отвлечься на их решение во время спринта. Полезнее всего в таком случае дать четкое определение проблемы, например: «Приоритет 1, серьезность 1. Ошибки, связанные с соглашением об уровне услуг (SLA), разрешено устранять в первую очередь».

Применяйте ретроспективы для более точного планирования поддержки продукта. Планирование спринтов на основании скорости выполнения работ и ретроспективы могут помочь команде измерить время, которое можно выделить при каждом спринте на поддержку. Когда команда проводит анализ проблем, встретившихся в течение спринта, ей нужно сопоставить количество времени, выделенное на поддержку продукта, и количество

времени, потраченное на нее в реальности, и исходя из этого составлять дальнейшие планы.

Разрешайте командам организовывать структуру на свое усмотрение. У разных команд возникает разное количество проблем, связанных с продуктом, новый объем работ может отличаться как в приоритете, так и в срочности, а у членов команды может быть разный уровень опыта и способностей в области решения проблем, возникающих с продуктами, над которыми они работали ранее. Все эти факторы говорят в пользу того, что лучше всего позволить командам вести поддержку продукта по-своему.

Agile-команды как черные ящики

Скрам открыто рассматривает команду как своего рода черный ящик. Если вы руководитель, то можете видеть ресурсы, которые затрачивают ваши команды, и то, с какой отдачей они работают, однако внутренняя работа команды вас не особо должна волновать.

В начале каждого спринта команда заявляет, какой объем работ она обязуется выполнить (ставит цель спринта). Таким образом, команда должна выполнить работу, не важно какую, к концу спринта. В течение спринта команду воспринимают как черный ящик — никто на протяжении спринта не вмешивается в ее работу и не добавляет новых задач. В конце спринта команда доставляет функционал, который обязалась реализовать в его начале. Спринт длится недолго, следовательно, менеджерам не придется долго ждать, чтобы проверить, справляется ли команда со своими обязательствами.

Сравнение команды с черным ящиком является преувеличением, подчеркивающим основную мысль. Отталкиваясь от сотен разговоров с менеджерами, я считаю, что такое восприятие команд приводит к более здоровому и эффективному менеджменту. Менеджерам не стоит анализировать каждую мелочь рабочего процесса — важно сосредоточиться на обеспечении у команды четкого представления о том, в каком направлении двигаться, а также на том, чтобы команда не сходила с намеченного пути. Им не нужно знать о каждом мелком решении или ошибках, которые допускают команды при продвижении в направлении к своей цели. Чрезмерная озабоченность мелочами противоречит ряду ключевых принципов, таких как декриминализация ошибок и обеспечение максимальной автономии команды.

Соответственно задачи руководителя в отношении такого черного ящика включают в себя устранение преград (препятствий), защиту команд от нарушений течения спринта, коучинг посредством разрешения конфликтов, устранение конфликтов приоритетов, поощрение развития команды, вовлечение новых членов команды, рационализацию

организационной бюрократии и поощрение команды к тому, чтобы она осмысливала свой опыт и извлекала из него уроки.

Готова ли ваша организация создавать Agile-команды?

Один из антипаттернов Agile — внедрение Скрама без введения настоящего самоуправления в командах. Если руководство постоянно пустословит о самоуправлении, продолжая при этом вмешиваться в каждую мелочь и контролировать каждый шаг, внедрить Agile не получится. Организациям не следует внедрять Agile, до тех пор пока они не готовы, не желают и не стремятся к созданию и поддержке самоуправления в командах.

Прочие соображения

Географически распределенные команды

Географическая распределенность привносит некоторые трудности в обеспечение эффективности команд. Более подробно это обсуждается в главе 7 «Еще более эффективные географически распределенные команды».

Открытая планировка офиса

Особенность некоторых случаев внедрения Agile состоит в том, что организации стали отказываться от замкнутых пространств вроде кабинетов и переходить к открытой планировке офиса для повышения степени сотрудничества. Не рекомендую так делать.

Вопреки ожиданиям, исследование Гарвардского университета показывает, что открытая планировка сокращает личное общение примерно на 70% в сравнении с отгороженными помещениями [Джарретт, 2018]. Другое исследование, проводимое в течение нескольких лет, показало, что открытая планировка приводит к неудовлетворенности сотрудников, повышает уровень стресса, снижает производительность труда, понижает творческий потенциал, вредит концентрации внимания, создает отвлекающие факторы и снижает мотивацию [Конникова, 2014].

Некоторым командам нравится опен-спейс (и они комфортно себя чувствуют), но в большинстве случаев это не так. В действительности команды всегда упорно противились открытой планировке [Джаретт, 2013]. В одной из статей было написано: «Официально: офисы с открытой планировкой признаны глупейшей прихотью менеджмента за всю историю» [Джеймс, 2018].

В изданной в 1996 году книге Rapid Development («Быстрая разработка») я обобщил результаты исследований того времени, которые показали, что наибольший уровень производительности достигался в личных или

близких к тому (два человека) кабинетах [Макконнелл, 1996]. Современные исследования указывают на актуальность этих выводов.

Для достижения большей эффективности советую следующее:

- Личные или близкие к этому кабинеты и открытые пространства для командной работы.
- Размещение команд по отгороженным помещениям, открытые пространства для командной работы и комнаты концентрации для временного пользования специалистами.
- Отгороженные помещения и комнаты концентрации.
- Отсек с открытой планировкой и комнаты концентрации.

Во всех перечисленных случаях, за исключением первого, я видел почти повсеместное использование наушников и частой работы из дома. И то и другое являются признаками того, что коллектив не может достаточно сосредоточиться для успешного выполнения своих обязанностей.

Рекомендации

Изучайте

- Пересмотрите состав ваших команд. Обладают ли их члены достаточной компетенцией, необходимой для самостоятельного принятия большей части решений?
- Опросите команду, чтобы понять, как у них *на самом деле*организовано тестирование (а не так, как показано на структурной схеме организации). Ваши команды эффективны, самодостаточны и сами справляются с тестированием своего кода? Есть ли у них собственные тестировщики?

Приспосабливайтесь

- На базе обзора состава ваших команд, предложенного выше, проведите сравнительный анализ, с помощью которого можно выяснить, какие навыки необходимо развивать, для того чтобы организовать самоуправление в командах.
- Создайте план пересмотра состава ваших команд и/или развития у них недостающих навыков, так чтобы каждая команда могла самостоятельно принимать решения и развиваться в направлении подлинного самоуправления.
- Создайте план обеспечения того, чтобы тестировщики являлись неотъемлемой частью ваших команд разработчиков.

Дополнительные ресурсы

[Агина Воутер и др., 2019]. How to select and develop individuals for successful Agile teams: A practical guide. McKinsey & Company. В этом исследовании рассматривается польза многопрофильности команд, работающих по Agile. В том числе рассматривается многопрофильность на основе пятифакторной модели личности и на основе модели профессиональных предпочтений, включающей в себя ценности, характерные для Agile.

ГЛАВА 6. Еще более эффективная культура команды

При работе по Agile организации находят взаимосвязь между структурой и культурой команды. Переход к самоуправлению требует изменений в культуре команды, которые дополняют и поддерживают способность команд к самоуправлению.

В этой главе описаны составляющие культуры Agile на уровне команд. В главе 17 «Еще более эффективная культура организации» приводится дальнейший обзор культуры Agile на уровне организации.

Ключевой принцип: мотивируйте команды автономией, профессионализмом и целями

Большинство исследований производительности показало, что производительность больше, чем от остальных факторов, зависит от мотивации [Боэм, 1981]. Что касается разработки ПО, единственный вид мотивации, который имеет значение, — внутренняя мотивация. Компания, по сути, занимает место в сознании сотрудников, платит им деньги, для того чтобы они думали о том, о чем компания хочет, чтобы они думали. Внешняя мотивация не срабатывает, потому что нельзя заставить кого-то думать о чем-то, вы можете только создать обстоятельства, в которых они будут думать о вашей проблеме, потому что она интересна им самим.

В своей книге 2009 года «Драйв» [2] Дэниел Пинк предложил теорию внутренней мотивации, в основе которой лежат факторы автономии, профессионализма и целей. Теория мотивации Пинка подтверждает, что при работе по Agile командам нужно работать эффективно.

Автономия

«Автономия» означает способность принимать решения в повседневной жизни и в работе — что, когда и с кем это делать.

Автономия связана с доверием. Если специалист считает, что компания недостаточно ему доверяет в принятии решений, он будет считать, что в действительности никакой автономии нет. Работа, которую вы выполняете, для того чтобы развить у кросс-функциональных команд, работающих по Agile, способность принимать решения и наделить их полномочиями принимать их самостоятельно, также позволяет командам чувствовать себя автономными.

Таблица 6.1. Практические методы, способствующие или подрывающие автономию

Как поспособствовать автономии	Как подорвать автономию
Руководить, задавая направление (в соответствии с более широким взглядом и задачами организации)	Руководить, обращая внимание на каждую мелочь ведения работ
Придерживаться направления	Часто изменять направление
Собрать команду, обладающую всеми навыками для самостоятельной работы	Держать в тайне от команды знания, которые ей необходимы для самостоятельной работы Создавать не настоящие команды, а просто группы специалистов, организованные по матричному принципу
Позволить командам экспериментировать в изменении практических методов на основании проведенных ретроспектив	Настаивать на заданном процессе независимо от опыта команды
Позволить командам работать в том темпе, который они определяют сами	Указывать командам, с какой скоростью им работать
Передавать требования на заранее согласованных условиях	Выдвигать требования команде или отдельным ее членам напрямую
Не вмешиваться в работу команд с высокой производительностью, продвигать работу к специалистам	Часто расформировывать и переформировывать команды, заставлять людей работать
Позволить командам допускать ошибки и учиться на них	Криминализировать ошибки и наказывать за них

Профессионализм

«Профессионализм» означает желание учиться и становиться лучше. Идея заключается не в том, чтобы достичь определенного уровня компетенции, а в том, чтобы постоянно совершенствоваться. Особенно это важно для технических специалистов. Как я уже писал много лет назад в своей книге Rapid Development [Макконнелл, 1996], возможность роста для разработчиков всегда была лучшей мотивацией, чем продвижение, признание, доход, статус, уровень ответственности и прочие факторы, которые, казалось бы, важнее. Акцент Agile на обучении из собственного опыта будет поддерживать профессионализм в ваших командах.

Таблица 6.2. Практические методы, способствующие или подрывающие профессионализм

Как поспособствовать профессионализму	Как подорвать профессионализм
Позволить выделять время на ретроспективы	Противодействовать проведению ретроспектив
Поощрять изменения, вносимые в каждом спринте ради обучения и совершенствования	Запретить вносить изменения или организовать громоздкий процесс утверждения
Позволить техническим специалистам изучать новые области технологий	Ограничить технических специалистов только насущными потребностями бизнеса
Позволить выделять время на подготовку и профессиональное развитие	Требовать, чтобы все время было распределено на краткосрочные цели проекта, не давать времени для подготовки
Поддерживать проведение «дней инноваций»	Противодействовать проведению экспериментов
Поддерживать осмысленную практику, например Code Kata	Настаивать на строгой концентрации на задачах, не разрешать выделять время на личное совершенствование
Позволить членам коллектива переходить на новые специализации	Требовать, чтобы специалисты работали только в той области, в которой у них больше всего опыта

Цель

Иметь цель — значит понимать, почему работа, которую вы выполняете, значима. Что такое общая картина? Насколько то, над чем вы работаете,

более великое и важное, чем вы сами? Чем оно полезно вашей компании или обществу в целом? Акцент Agile на прямом контакте с клиентами будет поддерживать понимание цели в вашей команде. Акцент Agile на общей ответственности и отчетности в команде усиливает чувство товарищества, которое также поддерживает понимание цели в команде.

Таблица 6.3. Практические методы, способствующие или подрывающие цель

Как поспособствовать цели	Как подорвать
Обеспечить постоянный контакт технических специалистов и непосредственных клиентов	Ограничить прямое взаимодействие технических специалистов и клиентов
Обеспечить частый контакт технических специалистов с внутренними бизнес-структурами	Изолировать технических специалистов от бизнес-структур так, чтобы они взаимодействовали редко
Регулярно обсуждать общую картину, сложившуюся вокруг работы команды	Обозревать общую картину только на нечасто проводимых крупных собраниях
Обеспечивать соответствие информации происходящему в действительности	Подавать клишированную информацию, оторванную от действительности
Объяснять команде реальную важность ее работы для общества: «Наш дефибриллятор спас столько-то жизней в прошлом году»	Настаивать на том, что общая картина — дело руководства, а команде не обязательно что-то знать
Подчеркивать значение высокого качества работ для организации	Обсуждать только срочную финансовую выгоду компании и/или краткосрочные цели доставки продукта

Блестящее сочетание автономии, профессионализма и цели

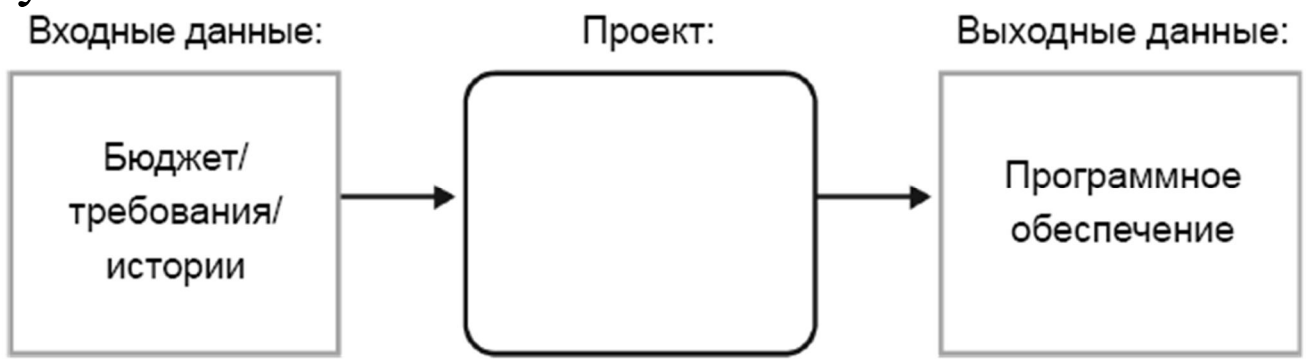
Исследование Дэниела Пинка показало, что команда, которая работает самостоятельно, понимает, почему она выполняет эту работу, и неуклонно совершенствуется при высоком уровне мотивации. Факторы, влияющие на создание эффективной команды, также позволяют создать мотивированную команду, и в этом взаимодополняющем сочетании эффективность и мотивация усиливают друг друга.

Ключевой принцип: развивайте мировоззрение, ориентированное на рост

Идея еще более эффективного Agile — цель, находящаяся в постоянном движении. Не имеет значения, насколько эффективно вы работали в этом году, в следующем нужно быть еще эффективнее. При этом, чтобы наблюдался рост, нужно позволять командам тратить время на совершенствование. Некоторые из таких улучшений должны происходить в регулярном цикле из ретроспектив и планирования спринтов, а некоторые — непосредственно во время спринтов.

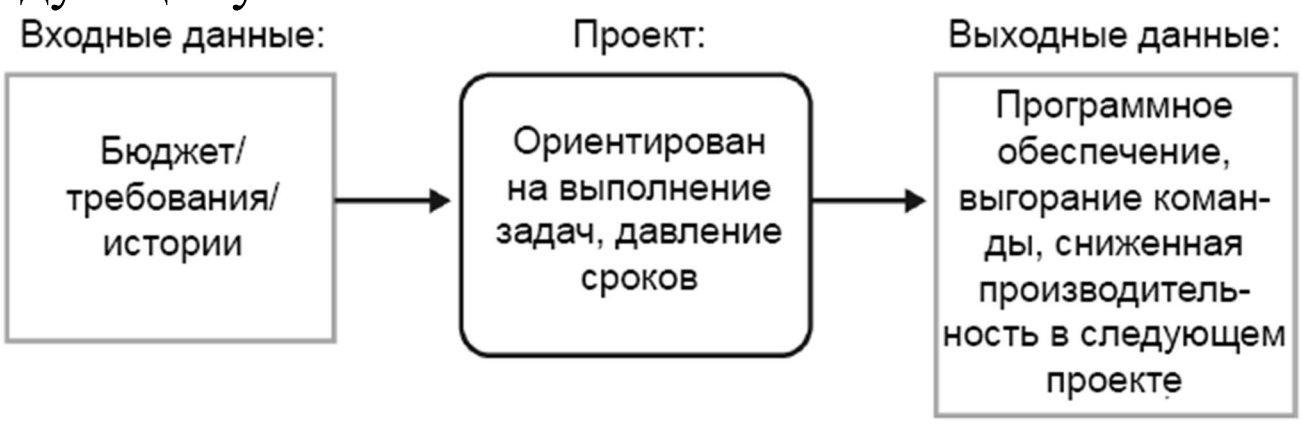
Чтобы становиться эффективнее, требуется мировоззрение, ориентированное на рост, которое можно охарактеризовать установкой «со временем мы можем стать лучше». А она есть не у всех руководителей.

Некоторые руководители смотрят на проекты по разработке ПО в категориях примитивной оценки затраченных усилий и полученных результатов:



С этой точки зрения единственной целью проекта является создание ПО, а единственный важный результат проекта — само программное обеспечение.

Более целостное представление о затратах на проект и его результатах рассматривает способности команды до и после проекта. Проект, ориентированный исключительно на выполнение задач, в котором, как правило, присутствует порция давления сроков, может приводить к следующему:



Если руководитель не ориентирован на рост команды, то легко можно допустить, что команда будет выдыхаться, а ее способности будут хуже, чем в начале проекта. Та же логика подходит к спринтам и релизам. В некоторых командах при работе по Скраму проявляется «усталость от спринта», когда спринт проходит в непостоянном темпе.

Разница между тем, как команда начинает вести проект и как она заканчивает проект, существенно влияет на эффективность организации. Во многих организациях каждый проект — это марш-бросок.

Проекты сосредоточены исключительно на задачах переднего плана, поэтому никогда нет времени на развитие специалистов в том направлении, в котором они хотят. На самом деле постоянное давление сроков буквально ухудшает показатели — с точки зрения их чувства автономии, профессионализма и, в конце концов, с точки зрения мотивации.

Это приводит к предсказуемым тенденциям, к тому, что команда сталкивается с выгоранием, лучшие специалисты команды уходят в другие компании, а потенциал организации со временем снижается.

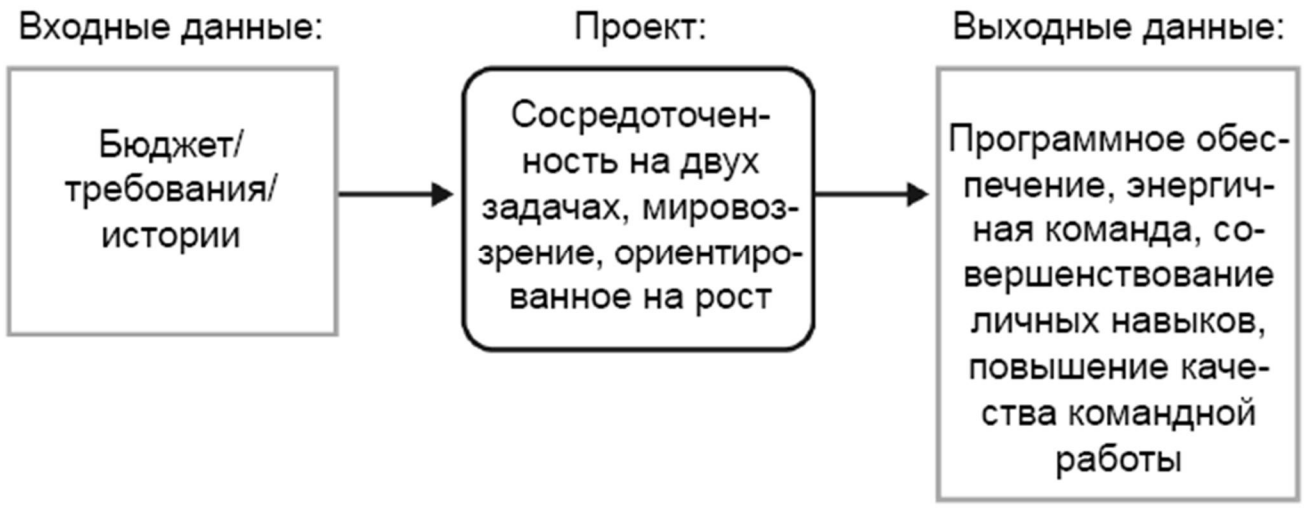
Организация, которая стремится быть эффективнее, будет рассматривать цель проектов по разработке ПО с позиций более комплексного

мировоззрения, ориентированного на рост. Конечно, создание функционирующего ПО является одной из целей проекта, но другая его цель — развитие способностей команды, которая производит это программное обеспечение. Идея такова: «Со временем мы сможем работать лучше, и мы будем находить время на достижение этой цели».

Мировоззрение, ориентированное на рост, приносит организации несколько преимуществ:

- Повышение энергичности членов команды.
- Улучшение личной и командной мотивации.
- Бˆольшая сплоченность команды.
- Повышение лояльности к компании (уменьшение утечки кадров).
- Улучшение технических и нетехнических навыков — улучшение качества кода и продукта.

Компания, которая понимает возможности получения выгоды из мировоззрения, ориентированного на рост, будет вести проекты следующим образом.



Традиционная для Agile мантра, гласящая об «устойчивых темпах», — это один из необходимых элементов эффективного применения Agile, но это спасает лишь от выгорания, а не гарантирует того, что команда будет непрерывно становиться лучше. В основе мировоззрения, ориентированного на рост, лежит фундамент, созданный устойчивыми темпами работы. Приверженность такому мировоззрению приносит дополнительные преимущества компании и ее сотрудникам.

Развитие способностей команды — одна из главных обязанностей руководителя в области разработки ПО. В главе 8 «Еще более эффективные сотрудники и их взаимодействие» описан системный подход к развитию способностей специалистов.

Ключевой принцип: развивайте ориентацию на бизнес

В области разработки ПО нет универсальных средств, но есть один практический бизнес-ориентированный метод, который достаточно редко применяется. Метод прост, а его преимущества значительно перевешивают трудности при внедрении.

Что же это за метод? Суть в том, чтобы просто наладить прямой контакт каждого разработчика с непосредственными клиентами, непосредственными пользователями продукта.

Некоторые компании сопротивляются налаживанию контакта с пользователями, поскольку боятся, что разработчики не способны на это. Они относятся к владельцу продукта (а также к менеджеру по продажам или бизнес-аналитику) как к защитному экрану между разработчиками и пользователями. Это и ошибка, и значительная упущенная возможность.

Для разработчика непосредственный контакт с пользователями часто представляется переломным опытом. Разработчик, который ранее выступал за техническую чистоту (что бы это ни значило) и рассматривал пользователей главным образом как источник надоедливых запросов о реализации нелогичных функций, становится ярым сторонником простоты использования и удовлетворенности пользователя.

Руководители бизнеса, которые делают доступным общение разработчиков и пользователей, неизменно сообщают, что преимущества, полученные благодаря пониманию взгляда пользователя на продукт, намного перевешивают любые риски, которые могут возникнуть. Технические специалисты развивают понимание того, как их труд выглядит на практике, как их пользователи зависят от них, чтˆо расстраивает пользователей и насколько велико может быть значение их работы, когда она действительно отвечает потребностям пользователей. Существует тесная взаимосвязь между коммуникацией разработчиков с пользователями и целью в контексте автономии, профессионализма и цели. Этот практический метод обеспечивает выгоды и от качества продукта, и от мотивации.

Вот несколько способов установить связь разработчиков и пользователей:

- Предложить разработчикам периодически прослушивать телефонные разговоры службы поддержки в течение нескольких часов подряд за один раз.
- Предложить разработчикам поработать в службе поддержки в течение нескольких часов.
- Отправить разработчиков наблюдать за использованием их ПО в деле.
- Попросить разработчиков понаблюдать за пользователями в UX-лаборатории через одностороннее стекло или монитор.
- Предложить разработчикам сопровождать менеджеров по продажам во время посещения клиентов или послушать телефонные переговоры.

Эти методы не рассматриваются как поощрение и наказание, это способ поддержки здорового бизнеса. Они подходят и для джунов, и для сеньоров, и для новичков в команде — для всех.

Важно, чтобы контакт с пользователем был регулярным, а не разовым. В противном случае разработчики могут чрезмерно зациклиться на

проблемах, которые наблюдали только в одном случае. Постоянное наблюдение необходимо для того, чтобы сформировался сбалансированный взгляд на проблемы пользователей.

Роль владельца продукта — слабое звено во многих организациях. Хотя развитие бизнес-мышления у технических специалистов не заменит хорошего владельца продукта, оно может смягчить положение при недостаточной его компетенции.

Налаживание прямого контакта разработчиков и пользователей — невероятно простая идея, которая практикуется очень редко, зато дает значимые результаты, если ее воплощают.

Прочие соображения

Навыки личного взаимодействия

Способность успешно работать в команде у специалистов зависит от их навыков личного взаимодействия. Подробнее см. в главе 8.

Личные установки и роли

Команды, как правило, работают лучше, когда соблюдается баланс между личными установками и ролями. Теория Белбина о ролях в команде представляет собой интересный и полезный способ оценки наличия ролей в команде. Теория включает в себя оценку того, как каждый специалист ведет себя в команде, насколько вероятно, что группа людей сможет хорошо сработаться вместе, и как отобрать кандидатов на каждую роль. Роли, по Белбину, включают в себя роли реализатора, координатора, генератора идей, мотиватора, разведчика, председателя, аналитика и доводчика.

Исследование IT-команд показывает высокую степень взаимосвязи между балансом ролей в команде и производительностью [Твярдохлеб, 2017].

Рекомендации

Изучайте

- Просмотрите списки, приведенные в табл. 6.1–6.3. Как вы оцените личное взаимодействие в командах относительно данных, приведенных в таблицах?
- Как вы оцените уровень взаимодействия в остальных частях вашей организации по спискам, приведенным в табл. 6.1–6.3?
- Предложите командам оценить свою мотивацию и моральный дух в начале и конце каждого проекта или цикла релиза. Что показывают цифры? То, что команда растет, продвигаясь устойчивыми темпами, или то, что она выгорает?

Приспосабливайтесь

- Измените ваши собственные действия, насколько это того требует, чтобы обеспечить автономию вашим командам.
- Внесите другие изменения, основываясь на обзоре табл. 6.1–6.3.
- Создайте план для того, чтобы обеспечить вашим командам более здоровое состояние и развитие бˆольших способностей к концу проекта, чем у них было в начале. Донесите командам пожелание о том, чтобы во время каждого цикла они тратили больше времени на обучение.
- Создайте план по обеспечению прямого контакта между разработчиками и клиентами.

Дополнительные ресурсы

[Дэниел Пинк, 2009]. Драйв. Что на самом деле нас мотивирует. (М.: Альпина Паблишер, 2013). В этой популярной книге о бизнесе выдвинута теория мотивации на основе автономии, профессионализма и цели, описанная в этой главе.

[Стив Макконнелл, 1996]. Rapid Development: Taming Wild Software Schedules. Несколько глав в этой книге так или иначе обсуждают мотивацию.

[Михал Твярдохлеб, 2017]. Optimal selection of team members according to Belbin’s theory. Научные журналы Морской академии в Щецине. 15 сентября 2017 г. В этой учебной статье обобщена теория Белбина о ролях в команде применительно к студенческим проектам. Твярдохлеб установил, что отсутствие хотя бы одной роли в команде приводило к тому, что команды были не в состоянии выполнять свои задачи.

[Кэрол Двек, 2006]. Mindset: The New Psychology of Success. Это классическое описание мышления, ориентированного на рост, которое включает в себя обсуждение того, как такое мышление можно применить к студентам, родителям, руководителям, романтическим партнерам и в других ролях.

ГЛАВА 7. Еще более эффективные географически распределенные команды

За более чем 20 лет работы с компаниями, где были географически распределенные команды разработчиков, доводилось наблюдать лишь небольшое количество случаев, когда производительность была сравнима с командами, сосредоточенными в одном месте. Мы не видели никаких признаков того, что географически распределенные команды, работающие по Agile, будут когда-нибудь столь же эффективны, как команды, члены которых находятся вместе. Однако в наши дни распределенные команды являются данностью для многих компаний, поэтому в этой главе будет рассказано о том, как добиться от них наилучших результатов из возможных.

Ключевой принцип: обеспечьте более тесную обратную связь

Один из принципов эффективности разработки ПО — обеспечение максимально возможной обратной связи. Многое из того, что описано в этой книге, можно вывести из этого принципа. Зачем нам нужен владелец продукта в команде, работающей по Agile? Чтобы обеспечить тесную обратную связь в отношении требований. Зачем нашим командам кросс-функциональность? Чтобы обеспечить тесную обратную связь в отношении принятия решений. Почему мы определяем и доставляем требования малыми партиями? Для того чтобы обеспечить тесную обратную связь между определением требований и программным обеспечением, которое можно запустить и показать. Зачем мы ведем разработку с тестированием в начале? Для того чтобы обеспечить тесную обратную связь между кодом и тестом.

Тесная обратная связь становится крайне важной при работе в контексте Cynefin «запутанные», потому что работу нельзя распланировать заранее, ее нужно прояснять с помощью многочисленных циклов *исследуйте*.*осознайте*.*реагируйте*. Такие циклы представляют собой вид обратной связи, которая должна быть как можно более тесной.

В географически распределенных командах наблюдается эффект ослабления обратной связи. Это замедляет принятие решений, увеличивает количество ошибок и необходимых исправлений, снижает эффективность и, в конце концов, приводит к срыву сроков проекта. Любое не личное общение с большей вероятностью будет приводить к недопониманию, а это ослабляет обратную связь. Различия в часовых поясах также ведут к задержкам реакции с таким же эффектом. Выполнение работы крупными партиями перед отправкой удаленной команде, например во время визита штатного владельца продукта к удаленной команде для поддержания личного общения, также ослабляет обратную связь. Добавьте к этому различия в языке, национальной культуре, культуре труда и издержки часовых поясов, вызывающих усталость, которая накапливается из-за участия в дистанционных собраниях в неудобные часы, — обратная связь ослабевает, а ошибок возникает больше.

Нам доводилось работать с компанией, в которой удаленная команда значительно уступала в производительности штатной команде. Когда мы перевели несколько удаленных сотрудников в штат, их производительность резко возросла за короткое время, пока они работали в штате, однако по возвращении домой их производительность снова упала. Это указывает на то, что проблема производительности возникла не из-за квалификации специалистов. Нехватка взаимодействия и задержки,

вызванные расстоянием в пол-экватора, приводили к невозможности удаленной команды работать эффективно.

Ослабленная обратная связь — наибольшая из проблем, которые мне встречались в распределенных командах. Она проявляется в разнообразных формах, как показано на рис. 7.1, все из которых я называю классическими ошибками:

- Разработка в одном месте, тестирование в другом.
- Владелец продукта в одном месте, разработчики в другом.
- Работа над общей функциональностью, которая разделена между двумя сторонами 50/50.

Рис. 7.1. Примеры того, как не стоит назначать обязанности в распределенной команде

Ни одна из этих конфигураций не дает положительных результатов — каждая провоцирует ситуацию, в которой люди, которым нужно часто общаться между собой, не могут быстро друг другу отвечать.

В начале 2000-х компании вели разработку и тестирование на разных площадках так, чтобы соблюдалась методология рационального использования солнечного времени (Follow the Sun , FTS) — тесты выполнялись, когда разработчики спали, чтобы сократить время выполнения задач. Логическим следствием, вместо ожидаемого результата, стало то, что разработчики могли не понимать отчетов об ошибках, а тестировщики — не понимать изменений, внесенных разработчиками. Получалось так, что решение вопросов, которое бы заняло у команды, сосредоточенной в одном месте, пару часов, занимало у удаленных команд полтора дня.

Лучшим практическим методом в этой области является создание команд, которые могут работать максимально автономно в каждой из точек, как показано на рис. 7.2. Выражаясь языком программного обеспечения, рассматривайте команды как обладающие высокой целостностью и слабой взаимосвязью.

Рис. 7.2. Пример того, как назначить обязанности в распределенной команде

Также не случайно, что лучший практический метод, подходящий распределенным командам, тот же, что и для команд в целом: создать самоуправляемые кросс-функциональные команды, у которых есть как способность, так и полномочия принимать необходимые решения на месте.

Навстречу успешной работе распределенных команд по Agile

Для успешной работы распределенных команд нужно следующее:

- График личного обсуждения вопросов.
- Улучшение материально-технической поддержки распределенных команд.
- Поощрение автономии, профессионализма и цели.
- Соблюдение закона Конвея (понятие рассматривается на с. 83).
- Отношение к командам Agile как к черным ящикам.
- Поддержание высокого уровня качества.
- Учет культурных различий.
- Изучение и приспособление.

Создание графика личного обсуждения вопросов

Большинство проблем при удаленной работе не технические, они относятся к области межличностного общения. Географическая отдаленность, разность часовых поясов, различия в языке, национальности и культуре, различия в культуре труда, статусе организаций снижают надежность передачи сведений и усложняют ее.

Важно периодическое личное общение. Как сказал один из топ-менеджеров одной транснациональной компании, «период полураспада доверия полтора месяца». Если вы видите, что ошибок становится больше, значит, пришло время посадить всех сотрудников в самолеты, отправить их играть, есть, пить вместе — проще говоря, развивать человеческие отношения.

Целью является то, чтобы часть сотрудников в течение приблизительно полутора месяцев путешествовала по различным рабочим площадкам, так чтобы 100% сотрудников посетили другие площадки в течение нескольких лет.

Улучшение материально-технической поддержки распределенных команд

Если вы хотите преуспеть в работе с распределенными командами, нужно вкладывать средства, силы и время.

График личного обсуждения вопросов. Организуйте обязательные встречи, на которых должны присутствовать все. Распределите неудобное для связи время равномерно по всем площадкам, чтобы ни на одной из них не возникало усталости из-за разности в часовых поясах. Предоставьте эффективные инструменты для удаленных встреч, а также хорошее сетевое соединение для использования этих инструментов. Настаивайте на полезных практических методах ведения встреч: создавайте повестки дня, определяйте задачи, не отклоняйтесь от темы, вовремя заканчивайте собрания и так далее.

Ситуативное общение. Поощряйте случайно возникающее общение между площадками. Предоставьте каждому члену команды технологии для связи: качественный микрофон, веб-камеру и надежное сетевое соединение. Предоставьте инструменты для текстового, срочного и потокового общения, а также доступ к сетевым форумам (Slack, Microsoft Teams и так далее).

Доверенные лица на местах. Назначьте на удаленных площадках доверенных лиц владельца продукта или менеджера по инжинирингу. Если команде не удается получить ответ от владельца продукта или инженера, которые работают удаленно, то она может обратиться к их доверенным лицам. Такие доверенные лица регулярно участвуют в личных обсуждениях со своими удаленными коллегами, поэтому их решения синхронизируются.

Ротация персонала. Подумайте о том, чтобы предложить сотрудникам переезд на постоянной или долгосрочной основе. Поскольку многие команды разработчиков состоят из специалистов разной национальности, нередко бывает такое, что некоторые члены команды хотят вернуться домой. Малоизвестный факт, что *Microsoft*наполнила свою первую площадку в Индии индийцами, которые к тому времени уже успели поработать в кампусе в Редмонде. Это помогло сформировать культуру компании и улучшить понимание особенностей индийской площадки.

Адаптация и обучение. Создайте график посещения удаленных площадок для новых сотрудников в качестве деятельности по адаптации. Предоставьте наставникам возможность проводить коучинг новых сотрудников по практическим методам для эффективной работы в условиях географической распределенности.

Поощрение автономии, профессионализма и цели

Некоторые компании распределяют команды равномерно по нескольким подразделениям, так чтобы у каждого из них был равный статус. Обычно компании предоставляют своим площадкам различный статус: штатная и удаленная, внутренняя и внешняя, родительская и дочерняя компания, главная площадка и подчиненная. Они распределяют различные виды работ по вторичным подразделениям, в том числе работу меньшей важности, а также ограничивают полномочия на этих площадках.

Разница в статусе и сниженный уровень автономии ограничивают мотивацию на каждой из площадок. Я обнаружил, что второстепенные команды склонны углубляться в себя и искренне оценивать свой статус и уровень ответственности. Менеджеры второстепенных команд часто сообщают о том, что их команды запрашивают больше автономии и самостоятельности, просят о возможности роста (профессионализм) и хотят понимать общую картину работы, которую они выполняют (цель).

Чтобы преуспеть в разработке, ведущейся на нескольких площадках, Agile и прочие методологии находят способы обеспечить каждую площадку работой, которую можно выполнять автономно, и позволить каждой площадке повышать свой профессиональный уровень. Активно доносите до каждой площадки то, почему их работа важна как для организации, так и для всего общества в целом.

Соблюдайте закон Конвея

Закон Конвея утверждает, что организации проектируют системы, которые копируют структуру коммуникаций в этой организации [Конвей, 1968] [3]. Такая структура включает в себя как формальную структуру управления, так и неформальное межличностное взаимодействие. Взаимодействие между этими структурами существенно в работе с географически распределенными командами.

Закон Конвея действует в обе стороны — техническое проектирование, в свою очередь, влияет на структуру коммуникаций в организации. Если команда распределена по трем площадкам, но техническая архитектура не предусматривает работы в трех независимых зонах, то командам придется испытать трудности, потому что они будут в технической зависимости от других команд, которые могут находиться за границей.

Если команда была распределена географически долгое время, техническая архитектура уже наверняка отражает структуру команды. Если ваша команда переходит на географическую распределенность, сравните техническую архитектуру и организацию коммуникаций и поищите несоответствия.

Команды как черные ящики

Как и в случае с совмещенными командами, восприятие команд как черных ящиков предполагает, что менеджеры будут себя вести скорее как дающие направление и не будут вмешиваться в работу команды по каждой мелочи. Управляйте затратами ресурсов и отдачей ваших команд. Избегайте контроля выполнения работы команд до мелочей.

Поддержание высокого уровня качества

Дисциплина Agile, предусматривающая постоянное поддержание разрабатываемого ПО в состоянии, близком к релизу, помогает не допустить значительного расхождения между командами, находящимися на расстоянии.

Отношение к каждой команде как к черному ящику гарантирует то, что выполненная этими командами работа будет высокого качества. Практический метод поддержания качества кодовой базы на уровне релиза требует высокой дисциплины, которой с трудом добиваются даже совмещенные команды.

Распределенные команды естественным образом склонны не так часто приводить код в состояние, готовое к релизу. Это ошибка. Географически распределенные команды рискуют разойтись в разных направлениях, если не будут этого осознавать, поэтому в целях управления рисками нужно как можно чаще сверять работу. А чтобы сверка кода проводилась эффективно, распределенные команды должны уделить пристальное внимание критериям готовности.

Усилия, требуемые для поддержания качества ПО на уровне релиза, привлекают внимание к издержкам географического распределения команд. Если распределенная команда обнаруживает, что тратит огромное количество времени на частую сверку для достижения качества релиза, решение заключается не в том, чтобы проводить сверку реже. Это повышает риск того, что команда не будет проводить ее вообще. Решение заключается в изменении методов совершенствования рабочего процесса, чтобы повысить надежность и частоту сверок. В некоторых случаях из-за трудностей сверки может быть принято решение сократить количество подразделений разработки.

Учет культурных различий

Распространенные культурные различия включают в себя:

- Готовность сообщить плохую новость, способность ответить «нет» хотя бы на простые вопросы.
- Отношение к руководству.
- Преобладание ценности индивидуальных или командных достижений.

- Ожидания продолжительности рабочего дня и баланс приоритетов между работой и личной жизнью.

Об этом уже много всего написано, и если вы не знакомы с этими проблемами, почитайте о них.

Изучение и приспособление

Разработка, в которой задействованы географически распределенные команды, затруднительна. Трудности будут различаться в зависимости от того, сколько у вас площадок, где они расположены, от архитектуры ПО, распределения работы по разным площадкам, а также способностей отдельных команд и специалистов на каждой площадке.

Чтобы географически распределенные команды заработали, командам требуется регулярно проводить ретроспективы и честно оценивать, что работает, что отнимает больше времени, чем нужно, и какие проблемы, которые относятся к работе в распределенной команде, ведут к снижению эффективности. Культурные различия могут создавать трудности в проведении ретроспектив, поэтому придется провести дополнительную работу и вдохновить сотрудников на честные разговоры.

На общесистемном уровне компания также должна поддерживать ретроспективы, которые специально нацелены на оптимизацию проблем, связанных с разработкой на разных площадках. Их результаты команды должны использовать для внесения изменений, которые направлены на устранение обнаруженных трудностей, при этом у команд должны быть полномочия на принятие таких изменений. В противном случае организация рискует получить низкую эффективность в географически распределенных командах.

Ненадлежащее руководство распределенными командами может снизить мотивацию как на основных, так и на вторичных площадках, что приводит к понижению морального духа и повышению текучести кадров.

Многие, а возможно даже большинство компаний, не могут достичь целей, ради которых было решено работать с географически распределенными командами. Нужно сделать очень многое и сделать это правильно, чтобы успешно работать с распределенными командами, и это не та область, где можно чем-то пренебречь.

Ключевой принцип: меняйте систему, а не людей

В географически распределенных командах возникает больший уровень недопонимания, что, в свою очередь, ведет к ошибкам. Такие команды тратят больше времени на устранение недочетов, чем совмещенные команды. Рост числа ошибок увеличивает уровень стресса, который приводит к большей вероятности перекладывания ответственности и поиска виноватых.

Чтобы успешно работать с географически распределенными командами, важно понимать силу принципа декриминализации ошибок.

Воспринимайте ошибки как проблемы системы, а не отдельных специалистов. Спросите: «Что в нашей системе могло привести к такой ошибке?» Это хороший метод, но особенно он важен в условиях географически распределенных команд.

Прочие соображения

Если вы сталкиваетесь с неэффективностью распределенных команд по той причине, что они не способны принимать решения на месте, определите, есть ли у вас подобные трудности с командами на основных площадках. Возможно, такая же проблема есть и там, просто она менее заметна, так как совмещенной команде проще компенсировать нехватку автономии.

Рекомендации

Изучайте

- Насколько у вас тесна обратная связь с распределенными командами? Допускаете ли вы какие-нибудь из ошибок, описанных в этой главе?
- Пересмотрите различия в языке, национальной культуре и культуре труда в подразделениях. Оцените влияние этих различий на возникновение ошибок, вызванных недопониманием.
- Организованы ли ваши команды так, чтобы у каждой из них были автономия, профессионализм и цель?
- Дисциплинированны ли ваши распределенные команды настолько, чтобы достаточно часто проводить сверку кода и обеспечивать качество на уровне релиза? По меньшей мере так же часто, как если бы они находились рядом?
- Систематически ли ваши команды используют принципы изучения и адаптации, так чтобы они могли научиться работать эффективнее в сложившихся условиях?

Приспосабливайтесь

- При необходимости проведите реорганизацию ваших команд и паттернов взаимодействия, чтобы обеспечить тесную обратную связь.
- Разработайте план улучшения взаимодействия и понимания между площадками.
- Разработайте план для поддержки автономии, профессионализма и цели в ваших распределенных командах.
- Разъясните вашим командам важность постоянного поддержания качества на уровне релиза и убедитесь в том, что они следуют соответствующим критериям готовности.
- Наделите ваши команды полномочиями вносить изменения согласно результатам ретроспектив.

Дополнительные ресурсы

Большая часть сведений в этой главе обобщает непосредственный опыт моей компании. Поэтому дополнительных ресурсов будет немного.

[Мелвин Конвей, 1968]. How do Committees Invent? Datamation. Апрель 1968 г. В этой работе впервые было введено понятие закона Конвея.

[Джон Хукер, 2003]. Working Across Cultures. Stanford University Press. В этой книге приведены общие соображения на тему работы в условиях культурных различий и присутствуют отдельные комментарии по Китаю, Индии, США и другим странам.

[Дженни Стюарт и др.]. Succeeding with Geographically Distributed Scrum. Серия Construx White Paper. Март 2018 г. В этом исследовании содержатся предложения для распределенных команд, характерных для Скрама. Там описан в основном тот же опыт, что я привел в этой главе.

ГЛАВА 8. Еще более эффективные сотрудники и их взаимодействие

В Манифесте Agile говорится, что люди и взаимодействие важнее процессов и инструментов. Но сейчас Agile гораздо больше сосредоточен на процессах, а не людях, а внимание, уделяемое человеку, ограничивается взаимодействием вокруг определенных структурированных видов совместных работ.

Ориентированное на рост мышление способствует обучению, но если стремление представляет собой не больше, чем общее желание, обучение будет проходить спонтанно и не принесет хороших результатов. Если вы согласны с той мыслью, что команда в конце проекта должна работать лучше, чем в начале, то необходимо выделить время на обучение и включить его в план.

В этой главе представлен системный подход к обучению технических специалистов и рассказано, какие области знаний наиболее важны и каких знаний чаще всего не хватает техническим специалистам. Поскольку эта глава отличается широким углом обзора, в конце ее вы найдете больше дополнительных ресурсов, чем обычно.

Потенциал от развития сотрудников

Максимальное наращивание эффективности людей должно стать краеугольным камнем любой программы, направленной на увеличение эффективности компании. За несколько десятилетий исследователи обнаружили, что производительность среди специалистов со схожим уровнем опыта разнится по меньшей мере в десять раз [Макконнелл, 2011]. Они также обнаружили, что производительность среди команд, работающих в одной отрасли, также разнится в десять или более раз [Макконнелл, 2019].

Такие различия в личной эффективности могут рождаться сами, а могут быть искусственны. Архитектора разработки облачных технологий Адриана Кокрофта однажды спросили, где он нашел таких удивительных людей. Он ответил ведущей компании *Fortune 500*: «Я нанял их у вас!» [Форсгрен, 2018]. Дело, конечно, в том, что хорошие результаты не возникают сами собой за одну ночь. Их добиваются со временем, а это значит, что если у компании есть желание стать эффективной, то она должна способствовать развитию специалистов. В интернете гуляет такая шутка:
Финдиректор: «Что будет, если мы вложимся в развитие наших сотрудников, а они потом уйдут?»
Гендиректор: «А представьте, что будет, если мы этого не сделаем, и они останутся?»
Поддержка в развитии ваших специалистов синергична во многих отношениях. Первая и основная причина для развития специалистов в том, что оно увеличивает вклад ваших сотрудников. Существует также синергия между методами «изучайте и приспосабливайтесь» при мышлении, ориентированном на рост, на уровне проекта и на уровне личного развития. Наконец, поддержка развития специалистов приносит вклад в мотивационную силу профессионального мастерства.

Как сообщается в масштабном исследовании технологических организаций с высокой производительностью Форсгрена, Хамбла и Кима: «В сегодняшнем быстро меняющемся мире, полном конкуренции, лучшее, что вы можете сделать для своей продукции, компании и сотрудников, — учредить культуру экспериментирования и обучения, вкладывать средства в развитие технической подготовки и управление так, чтобы в этом преуспеть» [Форсгрен, 2018].
Форсгрен, Хамбл и Ким также сообщили, что пригодность обстановки для обучения является одним из трех факторов, тесно связанных с производительностью доставки ПО.

В некоторых организациях возникает напряженность, когда сотрудник хочет изучить новую специализацию и отдалиться от своей прежней. Обычно это выглядит так: сотрудник желает сменить область деятельности, так как хочет развиваться, а компания не поощряет этого, требуя, чтобы сотрудник продолжал выполнять привычную работу, в которой он уже накопил достаточно опыта. Смена деятельности затруднена настолько, что наиболее мотивированные специалисты уходят в другие компании с целью дальнейшего профессионального развития.

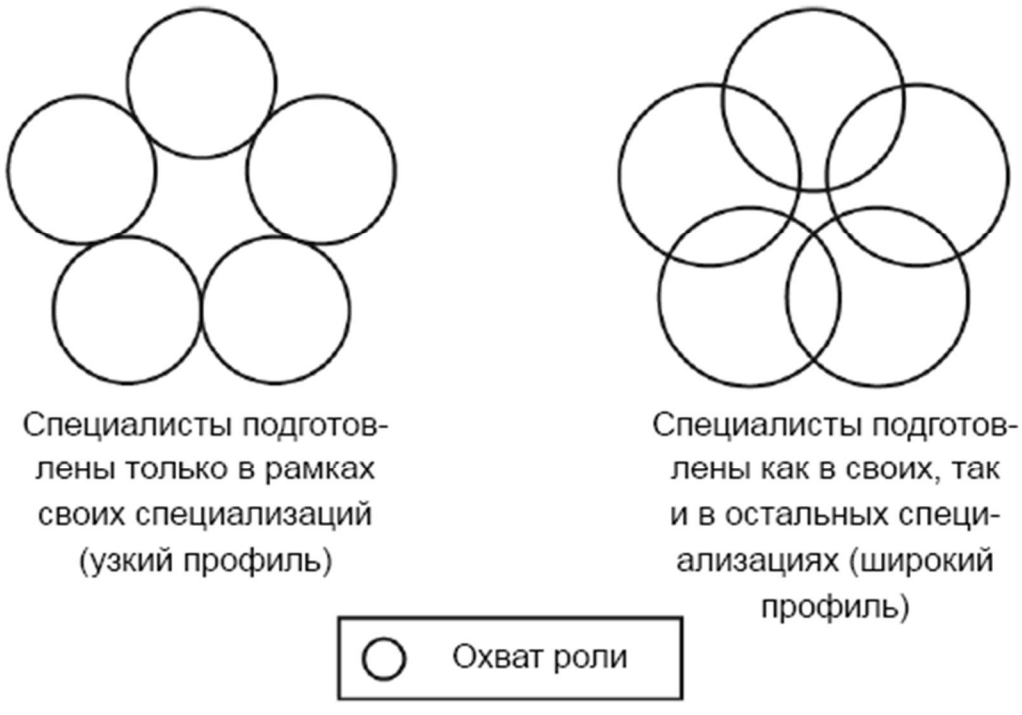
В организации, которая хочет развить эффективных специалистов, будет четко понятно, как пройти путь от младшего до старшего инженера, как добиться руководящей должности, как вырасти из технического руководителя до архитектора и так далее.

Ключевой принцип: развивайте способности команды посредством развития людей

Чаще всего историю карьеры профессионала можно сравнить с игрой в пинбол — прыгание из проекта в проект с разными технологиями и методологиями. Любой профессиональный опыт важен, но такой подход обрекает на спонтанное накопление разрозненного опыта и не дает систематической программы для постепенного обретения целостной компетенции и развития способностей.

Расширяйте роли

Кросс-функциональные команды зависят от наличия технических специалистов, которые хорошо справляются в своих специализациях и могут при необходимости выполнять другую работу. Под «расширением роли» я имею в виду способность специалиста работать в разных ролях. Сравните различия в широте роли ниже:



Какая команда больше всего подвержена текучке кадров? Какая более гибко распределяет работу? Какая лучше приспосабливается?

Софтверная организация, желающая добиться эффективности, будет поддерживать профессиональное развитие своих разработчиков тем, что будет создавать условия *для получения разнообразного опыта*, что, в свою очередь, позволит им добиваться высокого уровня профессионализма.

Развивайте три вида профессиональных способностей

Технические организации склонны ставить какую-либо область знаний выше других, считая ее самой важной для профессионала, однако это недальновидно. Высококомпетентные профессионалы должны иметь прочную базу в трех областях знаний:

- Технологические знания — знание конкретных технологий, например языков программирования и инструментов разработки.

- Практические методы разработки ПО — знание методов в областях, среди которых проектирование, написание кода, тестирование, требования и управление.
- Знание отрасли — знание отрасли, в которой ведется тот или иной бизнес, или области науки, в которой работает профессионал.

Техническим специалистам в разной мере требуется развитие в перечисленных областях знаний. Разработчику ПО необходимы глубокие познания в практических методах разработки и применения технологий, с меньшим упором на отрасль бизнеса или науки. Владельцу продукта требуется глубокое знание отрасли с меньшим упором на практические методы применения технологий и разработки ПО.

Специфику можно определить на основе распределения ролей.

Постройте структуру программы карьерного роста с помощью лестницы профессионального развития

Двадцать лет назад в нашей компании мы пришли к выводу, что программа карьерного роста для разработчиков была плохо определена и ей уделялось недостаточно внимания. Поэтому мы детально проработали лестницу профессионального развития, чтобы обозначить общее направление и обеспечить полную поддержку профессионального развития разработчиков ПО.

С тех пор мы продолжали поддерживать, обновлять и развивать эту лестницу, обеспечив свободный доступ ко многим материалам, которые помогут в карьерном росте, для разработчиков и организаций.

Лестница профессионального развития, разработанная *Construx*, помогает добиться долговременного карьерного роста разным специалистам по разработке ПО, в том числе разработчикам, тестировщикам, скрам-мастерам, владельцам продукта, архитекторам, бизнес-аналитикам, техническим руководителям и т.п. Лестница задает направление и структуру профессионального развития, при этом позволяя сотрудникам выбирать направление карьерного роста в соответствии со своими интересами.

Лестница профессионального развития состоит из четырех блоков:

- Области знаний в разработке ПО, основанные на стандартах, включая требования, проектирование, тестирование, управление качеством и так далее.
- Определенные уровни развития — начальный, компетентный и ведущий.
- Мероприятия по профессиональному развитию — в том числе подготовка, чтение и набор определенного опыта, необходимого для достижения уровня развития в каждой области знаний.
- План карьерного роста, характерный для той или иной роли, построенный на основе вышеперечисленного — областей знаний, уровней развития и мероприятий по профессиональному развитию.

В основе лестницы профессионального развития *Construx*лежит матрица профессионального развития 11 × 3, которая формируется из 11 областей знаний и 3 уровней развития [Макконнелл, 2018]. На рис. 8.1 изображена такая матрица.

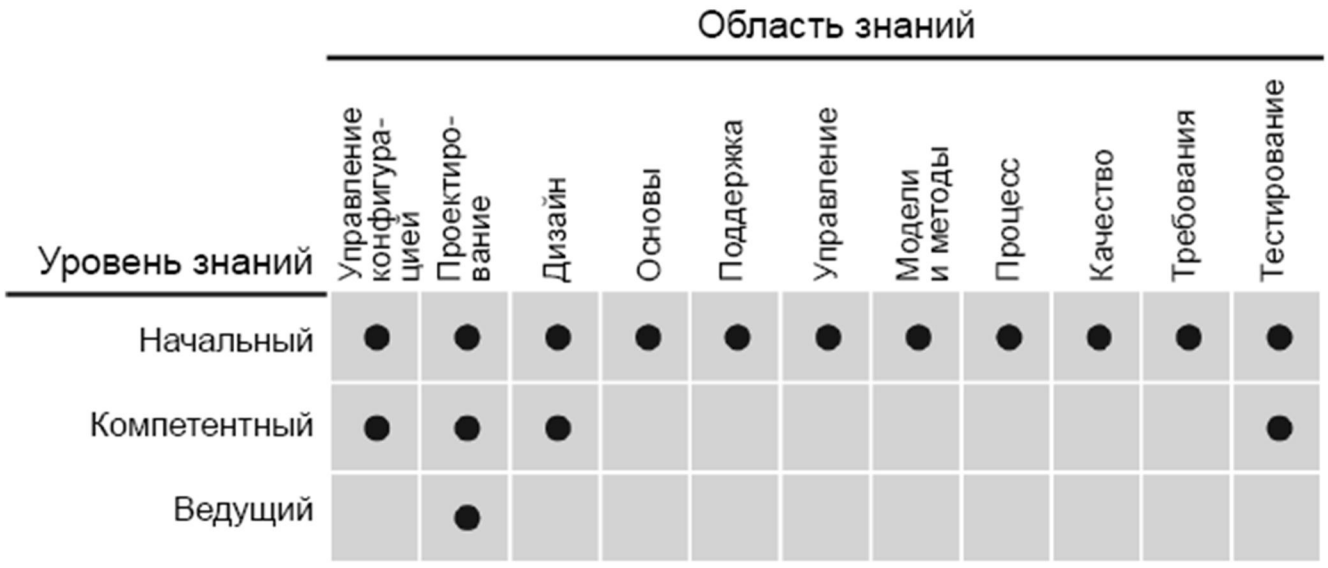


Рис. 8.1. Матрица профессионального развития размером 11 × 3

На этом примере в полях есть кружки, которые показывают, какие компетенции следует развивать специалисту, желающему достичь уровня старшего разработчика. Например, сеньору требуется получить уровень «ведущий» в области проектирования и уровень «компетентный» в управлении конфигурацией, дизайне и тестировании. Каждому кружку в матрице соответствует определенный набор материалов для чтения, подготовки и набора необходимого опыта для достижения указанного уровня развития.

Матрица профессионального развития выглядит просто, но это необычайно мощный инструмент. Цели карьеры можно определить с помощью полей, отмеченных в матрице. Карьерный рост можно определить с помощью графика, проведенного через отмеченные поля матрицы. Мероприятия по профессиональному росту можно запланировать в соответствии с полями матрицы в зависимости от целей.

Матрица, включающая в себя 11 областей знаний, основанных на стандартах, и 3 определенных уровня развития, представляет собой каркас для карьерного роста, одновременно обладающий четкой структурой, высокой степенью гибкости и хорошо подстраивающийся под актуальные потребности. Наиболее важно то, что четко прослеживается поэтапный план роста до высокого уровня.

Чтобы узнать подробнее о лестнице профессионального развития, в том числе о том, когда происходит развитие профессионализма, предложениях по поддержке профессионального развития и других вопросах реализации, советую прочитать мою работу Career Pathing for Software Professionals [Макконнелл, 2018].

Более эффективное взаимодействие в команде

В то время как каждая команда может улучшить показатели, благодаря тому что специалисты совершенствуются в разработке ПО, многим командам не удается добиться эффективности, из-за того что взаимоотношения между членами команды не складываются. Методология Agile требует личного взаимодействия, поэтому бесконфликтность в Agile важна больше, нежели в модели последовательной разработки.

По опыту работы я уверился в том, что некоторые гибкие навыки (soft skill) больше всего полезны для Agile-команд.

Эмоциональный интеллект

Если вы когда-либо наблюдали, как два разработчика развязывают в переписке войну из-за всяких технических мелочей, то это признак необходимости развития эмоционального интеллекта у членов команды.

Ценность эмоционального интеллекта важна для лидеров. Дэниел Гоулман сообщил журналу Harvard Business Review о том, что 90% разницы в производительности между успешными и средними командами можно обусловить нехваткой эмоционального интеллекта (также встречается сокращение EQ) [Гоулман, 2004]. Исследование, проведенное среди 500 самых востребованных соискателей, обнаружило, что эмоциональный интеллект гарантировал больший успех в трудоустройстве, чем собственно интеллект или опыт [Чернисс, 1999].

Технические специалисты получат пользу от расширения осознания своих собственных эмоциональных состояний и эмоциональных состояний других людей, улучшения регулирования своих эмоций и управления взаимоотношениями с другими.

Я считаю, что модель RULER, разработанная Йельским центром эмоционального интеллекта, станет полезным ресурсом в этой области [Йель, 2019]. Модель RULER предполагает:

- Признание (Recognizing) эмоции в себе и других.
- Понимание (Understanding) причин и следствий эмоций.
- Точное обозначение эмоций (Labeling).
- Выражение (Expressing) эмоций соответствующим образом.
- Эффективное регулирование (Regulating) эмоций.

Модель RULER изначально разрабатывали для работы с подростками, которых впоследствии приспосабливали к работе со взрослыми, особенно при работе со взрослыми в группе.

Взаимодействие с различными типами личности

Специалисты по продажам понимают, что разные люди общаются по-разному, поэтому выражают свои мысли исходя из того, с кем ведут разговор.

Техническим специалистам зачастую нужны явная инструкция и поощрение, чтобы их стиль общения соответствовал их аудитории.

Изучение типов личности помогает им понять, что разные люди по-разному смотрят на различные факторы при принятии решения (например, на данные или чувства). Они по-разному выражают мысли и эмоции, по-разному реагируют на стресс. Обозначение типов личности, наблюдение за тем, какие типы личности можно применить к окружающим, а также оценка своего типа личности часто открывают глаза специалистов на действительные проблемы.

Я думаю, что модель социальных стилей будет интуитивно понятным инструментом для понимания типов личности [Малквин, 2014]. Социальные стили основаны на наблюдениях за поведением, и вам не понадобится знать результаты чьего-то теста, чтобы понять, как найти общий язык. DISC [4], типология Майерс — Бриггс и психология восприятия цвета также могут принести пользу.

Ценность внимательного отношения к различиям в социальных стилях лучше всего проявляется в улучшении взаимодействия между людьми с разными типами личности. Как показано на рис. 8.2, согласно модели социальных стилей технические специалисты склоняются к аналитическому стилю, продажники — к экспрессивному, а менеджеры — к контролирующему (конечно, все это обобщения с многочисленными исключениями).

Изучение социальных стилей может помочь техническим специалистам эффективнее взаимодействовать со специалистами по продажам, помочь им лучше справляться с организационными задачами и улучшить взаимодействие между членами команды различных типов личностей. Некоторые технические специалисты воспринимают подстраивание стиля общения в зависимости от типа личности, с которой приходится взаимодействовать, как что-то нечестное. Это может стать самоограничением, препятствующим карьерному росту. Изучение различных стилей общения может стать полезным и помочь преодолеть подобное ограничение.

Рис. 8.2. Обзор модели социальных стилей

Научная обоснованность этих популярных моделей вызывает сомнение. Если вам интересен более научный подход, присмотритесь к модели Большой пятерки (она же OCEAN). Я придерживаюсь мнения, что все модели неправильны, но какие-то из них полезны. И полагаю, что модель социальных стилей будет особо полезной.

Критические разговоры

Структурный подход может хорошо работать для человека, у которого нет интуитивного ощущения, как выполнять задачу. Подход, называемый «критические разговоры», предназначенный для трудных разговоров, — эффективная модель, применимая в следующих ситуациях [Паттерсон, 2002]:

- Ставки высоки.
- Мнения расходятся.
- Эмоции бьют через край.

В техническом контексте критические разговоры могут возникать в самых разных ситуациях, например стычках между сотрудниками из-за проблем производительности, выбора подхода при проектировании, донесения плохих новостей стейкхолдерам и так далее.

Взаимодействие с топ-менеджерами

Понимание разных типов личности дает полезную основу для совершенствования взаимодействия в целом, и в частности с руководителями высшего звена.

Как написал один из рецензентов этой книги: «Голова полностью забита проблемой и нужен целый день, чтобы ее решить. У вашего начальника

есть всего семь минут и достаточно свободной рабочей памяти для определения трех ключевых пунктов».

Определение типа личности руководителя (по модели социальных стилей), понимание его хода мысли при принятии решения и способность предугадать его реакцию на стресс поможет всем техническим специалистам успешно наладить контакт.

Стадии развития команды

Модель стадий развития группы по Такмену практически стала ходовой в управленческих кругах. Разработка ПО так часто ведется в команде, а команды часто меняются, и поэтому членам команды полезно бы усвоить четыре стадии по Такмену: формирование, конфликт, нормирование, функционирование (рис. 8.3).

Рис. 8.3. Модель стадий развития группы по Такмену

Я обнаружил, что команды, находящиеся на стадии формирования и конфликта, при осознании того, что происходящее нормально, чувствуют облегчение.

Кроме того, такое осознание помогает им быстрее двигаться в направлении стадий нормирования и функционирования.

Лидерам следует понять, что это нормальный и ожидаемый процесс. Важно понимать, что роспуск и набор новой команды требуют времени, а также то, что каждый новый состав каждый раз проходит через все стадии до полноценного функционирования.

Усовершенствованные модели принятия решений

Команды разработчиков принимают огромное количество решений по поводу требований, приоритетов, подходов к проектированию, внесения изменений в процесс… Список бесконечен. Полезно знать несколько моделей принятия решений, ориентированных на команды. Мне удавалось успешно применять усовершенствованные практические методы принятия решений, в том числе голосование большим пальцем/римское голосование, голосование кулаком и пятью пальцами, точечное голосование и принятие решения лидером.

Проведение эффективных встреч

Стандартные встречи в Скраме хорошо структурированы: распределение ролей, постановка цели, основная повестка дня полностью регламентированы, что позволяет следить за ходом встреч и эффективно использовать время.

Другие виды встреч, проводимые во многих компаниях, чудовищно убивают время. Для общих собраний полезно составить руководство по эффективному проведению встреч. Как минимум оно должно содержать один стандартный совет: ставьте четкую цель встречи, ясно обозначайте ожидания относительно результата встречи. Делайте встречи короче, приглашайте только тех людей, которые помогут получить результат, объявляйте встречу завершенной сразу, как только будет достигнута ее цель, и так далее. Хорошим источником в этой области послужит How to Make Meetings Work [Дойл, 1993].

Мышление win-win для взаимодействия

Развитие мышления, основанное на том, как помочь окружающим добиться успеха, создает благоприятные тенденции внутри команды. Лучшая модель из мне известных — это Ротарианский четырехвопросник (Rotary International):

1. Соответствует ли это истине?
2. Честно ли это во всех отношениях?
3. Будет ли это способствовать взаимопониманию и укреплению дружеских отношений?
4. Выгодно ли это для всех заинтересованных сторон?

Каждое решение или взаимодействие, которое проходит через четырехвопросник, с большой вероятностью поможет построить более крепкую команду.

Общие навыки личного взаимодействия

Любой человек может получить пользу от периодического пересмотра своих навыков личного взаимодействия. Книга «Как завоевывать друзей и оказывать влияние на людей», написанная Дейлом Карнеги, — по-прежнему такое же хорошее руководство по эффективным взаимоотношениям, каким оно было почти 100 лет назад, когда проводилось соответствующее исследование [Карнеги, 1936].

Рекомендации

Изучайте

- Поразмышляйте о том подходе, который применяют у вас в компании для максимального увеличения личных возможностей. Предусматривает ли такой подход постоянное развитие каждого сотрудника после приема на работу?
- Пересмотрите время, выделяемое компанией для профессионального развития. Принимая в расчет количество выделяемого времени, реально оцените, насколько ваши команды могут продвинуться в профессиональном развитии?
- Опросите людей. Насколько важны для них четко определенные возможности профессионального роста? Насколько они удовлетворены поддержкой, получаемой от компании сейчас?
- Рассмотрите взаимодействие сотрудников в вашей организации, не связанное с техническими операциями. Насколько эффективно люди проводят встречи, сотрудничают, взаимодействуют с руководством и демонстрируют другие навыки взаимодействия?
- Поразмышляйте о конфликтах, которые возникали внутри команды, связанных с техническими и прочими операциями. Как бы вы оценили уровень эмоционального интеллекта у специалистов?

Приспосабливайтесь

- Регулярно составляйте план выделения времени на профессиональное развитие.
- С помощью лестницы профессионального развития от *Construx* (или любого другого подхода) обеспечьте наличие у каждого из членов коллектива определенной программы профессионального роста, которая бы представляла для них значимость.
- Создайте план улучшения навыков межличностного взаимодействия для членов ваших команд, включающий изучение типов личности, взаимодействие внутри организации, разрешение конфликтов и обеспечение взаимовыгодного сотрудничества.

Дополнительные ресурсы

[Дейл Карнеги, 1936]. Как завоевывать друзей и оказывать влияние на людей. Если прошло несколько лет с прочтения этой книги, обязательно прочтите ее еще раз. Вас удивит то, насколько актуальны советы, несмотря на их давность.

[Майкл Дойл и Дэвид Штраус, 1993]. How to Make Meetings Work! Это классическое описание того, как проводить встречи эффективно.

[Роджер Фишер и Уильям Юри, 2011]. Переговоры без поражения. Гарвардский метод (М.: Манн, Иванов и Фербер, 2018). Это классическая работа по достижению взаимовыгодного сотрудничества. Хотя в названии книги говорится о переговорах, на самом деле она отлично помогает решить проблемы в группе.

[Дэниел Гоулман, 2005]. Эмоциональный интеллект. Почему он может значить больше, чем IQ (М.: Манн, Иванов и Фербер, 2020). Это книга, в которой впервые обосновывается то, что значение эмоционального интеллекта не ниже, чем собственно интеллекта.

[Патрик Ленсиони, 2002]. The Five Dysfunctions of a Team. Это краткое руководство по бизнесу, описывающее в иносказательной форме хронику команды, деятельность которой находится в беспорядке, в котором затем приводится модель для создания и поддержания здоровых команд.

[Генри Липманович и Кейт Маккэнделс, 2013]. The Surprising Power of Liberating Structures. Эта новаторская книга описывает различные паттерны «освобождающих структур» взаимоотношений в группе.

[Стив Макконнелл и Дженни Стюарт, 2018]. Career Pathing for Software Professionals. Сетевой ресурс. В этом исследовании рассматриваются происхождение и структура лестницы профессионального развития *Construx*. Сопутствующие работы о реализации описывают путь карьерного роста до архитектора, менеджера по контролю качества, владельца продукта и технического руководителя.

[Керри Паттерсон и др., 2002]. Crucial Conversations: Tools for talking when the stakes are high. Это книга, которая легко читается и приводит убедительные доказательства в пользу того, что мир стал бы лучше, если бы каждый обладал навыками ведения критических разговоров.

[Ротари Интернэшнл, 2019]. Ротарианский четырехвопросник. Сетевой ресурс. Поиск в Сети выдаст большое количество описаний исторического и современного применения этого четырехвопросника. Статья в «Википедии» неплохо передает суть четырехвопросника.

[TRACOM Group, 2019]. Сетевой ресурс. На сайте TRACOM содержится много материалов по модели социальных стилей, в том числе обзор описаний модели, доклады о ее достоверности и сравнение модели социальных стилей с другими известными моделями, такими как типология Майерс — Бриггс.

[Wilson Learning, 2019]. Сетевой ресурс. На сайте Wilson Learning находится несколько статей по модели социальных стилей, в которых больше рассказано о применении модели в продажах (модель социальных стилей TRACOM и Wilson Learning одинаковы для неформальных практических целей).

[Йельский центр эмоционального интеллекта, 2019]. Модель RULER. Сетевой ресурс (2019 год). Описываются модель RULER и ее применение, особое внимание уделяется применению модели в образовательных целях.

ЧАСТЬ III. Еще более эффективная работа

В этой части подробно описывается рабочий процесс в Agile-проекте. Обсуждается, как организована работа в крупных проектах, а также сопутствующие проблемы. Рассматриваются конкретные виды работ, в том числе работа по контролю качества, требованиям и доставке ПО.
Если вам более интересны проблемы топ-менеджмента, чем подробный разбор проблем рабочего процесса, переходите сразу к части IV «Еще более эффективная организация». Если в компании возникают проблемы с крупными проектами, прочитайте главу 10 «Еще более эффективное выполнение крупных проектов», прежде чем перейти к части IV.

ГЛАВА 9. Еще более эффективное выполнение проектов

В предыдущей главе мы рассмотрели, как организовывать и поддерживать разработчиков при работе по Agile. В этой главе обсуждается, как правильно организовать и поддерживать процесс разработки при работе по Agile.

Большую часть разработки ПО организуют в проекты. В организациях используют самые различные термины для обозначения понятий, связанных с проектом, в том числе «продукт», «программа», «релиз», «цикл релиза», «функция», «поток создания ценности», «рабочий поток» и прочее.

Терминология значительно различается. Некоторые компании считают, что понятие «релиз» — это синоним слова «проект». Другие думают, что понятие «релиз» относится к последовательной разработке, поэтому прекратили его использовать. Третьи определяют понятие «функция» как объем работ, рассчитанный на 3–9 человек и 1–2 года. В этой главе под словом «проект» я буду иметь в виду любые из перечисленных видов работ, то есть работу некоторого количества сотрудников в течение длительного времени.

Ключевой принцип: небольшие размеры проектов

За последние 20 лет наиболее известны случаи успешного применения Agile в небольших проектах. Agile первые 10 лет своего существования уделял большое внимание тому, чтобы проекты были небольшими, то есть 5–10 человек в команде (например, 3–9 разработчиков, владелец продукта

и скрам-мастер). Акцент на небольшом размере проектов очень важен, потому что с небольшими проектами успешно справиться легче, нежели с крупными, как показано на рис. 9.1.

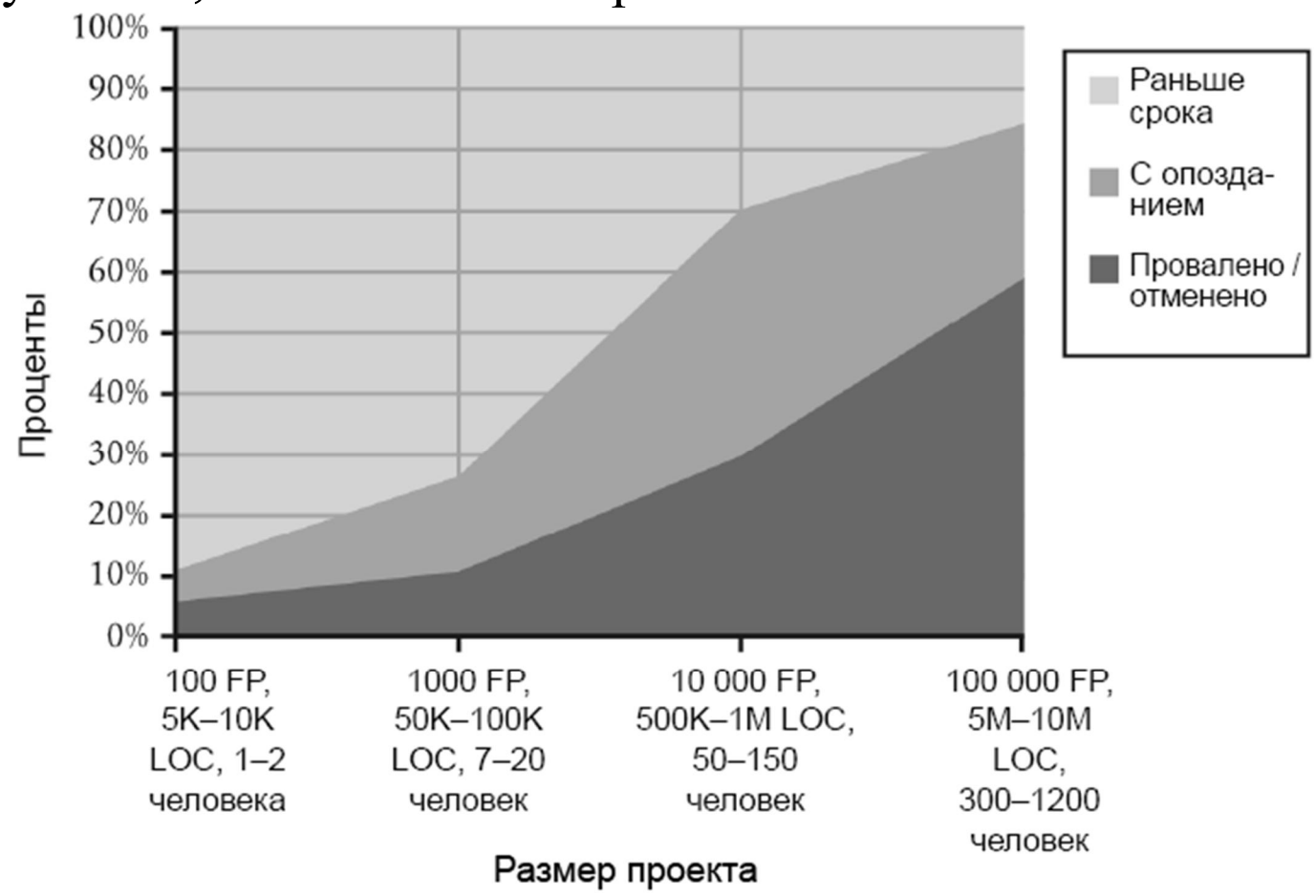


Рис. 9.1. Чем больше проект, тем выше вероятность сорвать сроки, не уложиться в бюджет и выше риск провала (Джонс, 2012). Проект можно измерять в единицах функций (FP), а также в строках кода (LOC). Сравнения размеров, приводимые в единицах функций, строках кода или размеров команд, приблизительны

Каперс Джонс уже 20 лет постулирует, что мелкие проекты чаще завершаются успешно, чем крупные [Джонс, 1991; 2012]. Я обобщил большинство исследований о зависимости успешности проектов от размера в своих книгах «Совершенный код» [Макконнелл, 2004; СПб.: Питер, 2007] и Software Estimation: Demystifying the Black Art [Макконнелл, 2006].

Небольшие проекты благополучнее по нескольким причинам. Крупные проекты требуют вовлечения большего количества специалистов, и число взаимосвязей между людьми в командах и между самими командами увеличивается в нелинейной прогрессии. А чем более сложными становятся взаимоотношения, тем больше ошибок совершается при взаимодействии. Ошибки взаимодействия приводят к ошибкам в требованиях, проектировании, написании кода — в общем, к другим ошибкам.

Следовательно, чем крупнее становится проект, тем больше допускается ошибок (рис. 9.2). Это говорит не просто о том, что общее количество ошибок растет, а о том, что в крупных проектах несоизмеримо больше ошибок!

Чем выше частота появления ошибок, тем ниже эффективность стратегий обнаружения недочетов. Это означает, что количество недочетов в готовом продукте непропорционально возрастает.

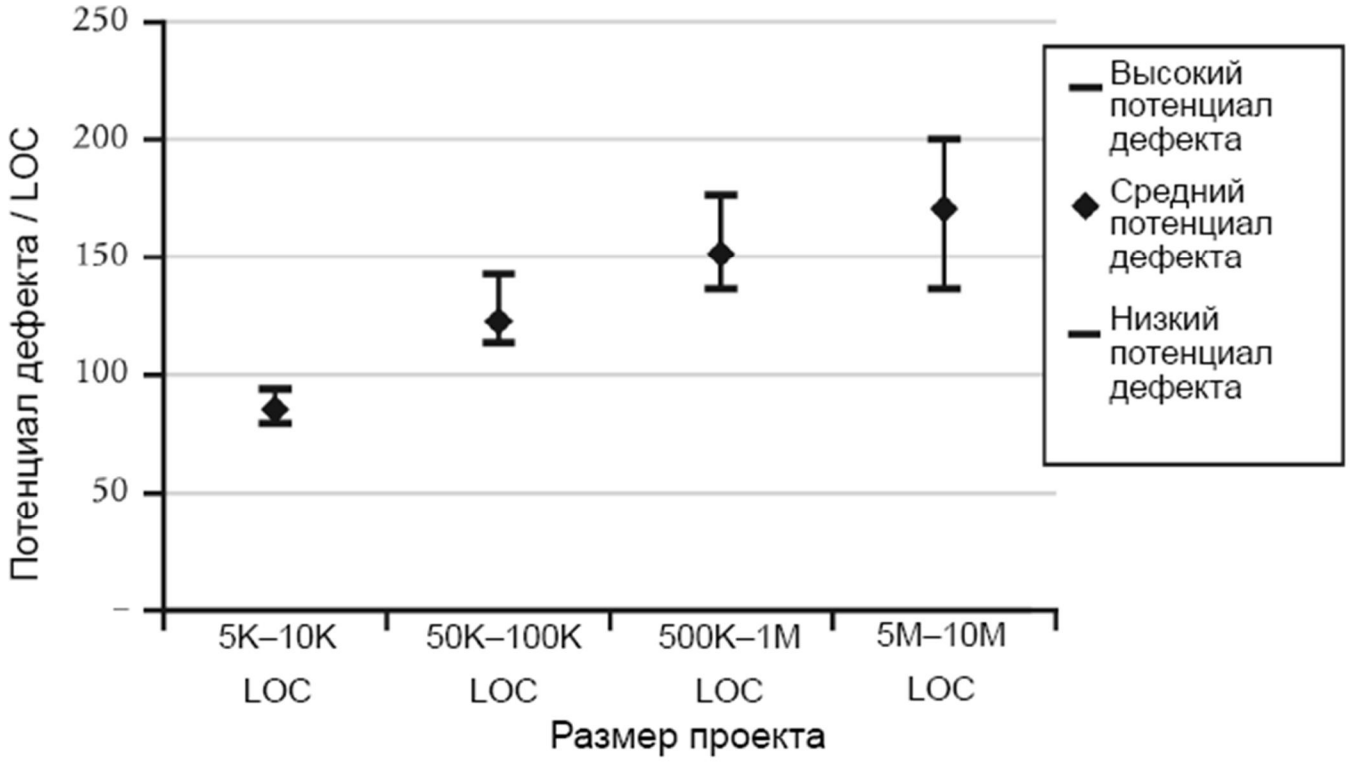


Рис. 9.2. Чем крупнее проект, тем больше частота появления ошибок (потенциал дефекта). Адаптация исследований [Джонс, 2012]

Требуется также больше усилий для устранения ошибок. Следовательно, как показано на рис. 9.3, более мелкие проекты отличаются большей производительностью на человека, но чем больше размер проекта, тем сильнее падает производительность. Это явление известно как «издержки масштаба».

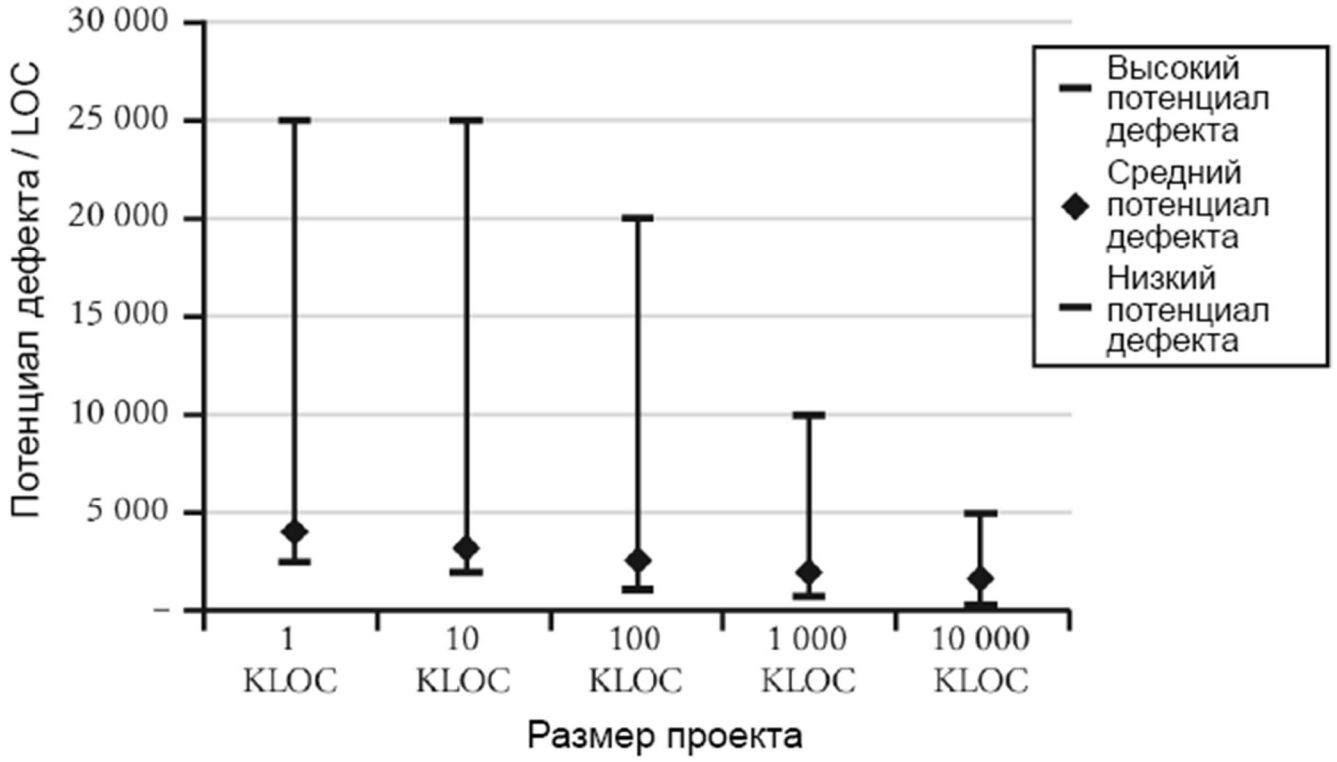


Рис. 9.3. Чем крупнее проект, тем ниже производительность на человека. Адаптация исследований [Макконнелл, 2006]

Эту обратную зависимость между размером и производительностью тщательно исследовали и проверяли на протяжении более 40 лет. Фред Брукс рассуждал на тему издержек масштаба при разработке ПО в первом издании своей книги «Мифический человеко-месяц» [Брукс, 1975; СПб.: Питер, 2021]. Работа Ларри Патнэма по оценке издержек при создании ПО подтвердила наблюдения Брукса [Патнэм, 1992]. Исследование модели издержек разработки (Cocomo) эмпирически подтвердило существование издержки масштаба дважды — в исследовании конца 1970-х годов и в обновленном исследовании конца 1990-х [Боэм, 1981; 2000].

Вывод таков: чтобы максимально увеличить вероятность успешного выполнения проекта по Agile, старайтесь сохранять небольшие масштабы проекта (и команды).

Конечно, невозможно сделать так, чтобы каждый проект был мелким. В главе 10 «Еще более эффективное выполнение крупных проектов» изложены подходы к крупным проектам, в том числе предложения о том, как их уменьшить.

Ключевой принцип: короткие спринты

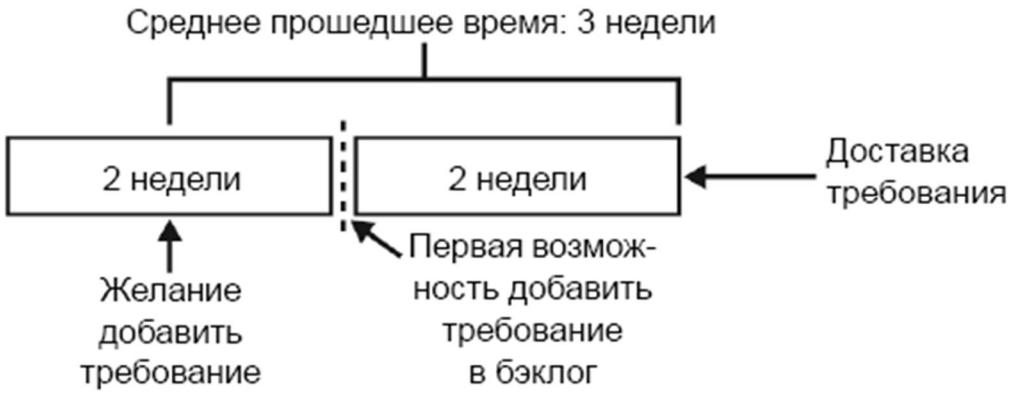
Естественный вывод из того, что предпочтительны небольшие размеры проекта, — такой, что спринты не должны длиться долго. Можно подумать, что небольшой проект уже сам по себе залог успеха. Но короткие спринты в 1–3 недели способствуют всестороннему успешному ведению проектов. Это описано в следующих нескольких разделах.

Короткие спринты снижают количество промежуточных требований и улучшают реагирование на новые требования

В Скраме новые требования можно добавлять между спринтами. Но когда спринт начался, до следующего спринта требования добавлять уже нельзя. Это разумно, если спринт длится 1–3 недели.

Если циклы разработки длятся дольше, то стейкхолдеры по ходу работы все настойчивее просят добавить требования, поэтому просьбы отложить добавление требований уже не так обоснованны. Если цикл при последовательной модели разработки длится полгода, то попросить стейкхолдера отложить реализацию нового требования до следующего цикла означает, что работа над требованием начнется только в следующем цикле — то есть в начале цикла это требование только добавят, а потом нужно ждать доставки продукта в конце этого цикла. В среднем это занимает 1,5 цикла, то есть 9 месяцев.

В противоположность этому обычные для Скрама спринты длятся две недели. Это означает, что стейкхолдеру, который желает добавить новое требование, придется подождать в среднем три недели.



Просить стейкхолдера подождать 9 месяцев для реализации требования зачастую неуместно. А три недели — почти всегда уместно. Это значит, что скрам-команда спокойно работает, не боясь появления новых требований посреди спринта.

Короткие спринты обеспечивают большую оперативность при работе с клиентами и заинтересованными сторонами

Каждый спринт представляет собой новую возможность для демонстрации рабочего ПО, проверки требований и проработки обратной связи со стейкхолдерами. При стандартных двухнедельных спринтах команды 26 раз в год дают себе возможность быть оперативнее! При трехмесячном же цикле разработке такая возможность представляется лишь четыре раза в году. Пятнадцать лет назад проект, рассчитанный на три месяца, считался коротким. Сегодня такой график означает, что вы не сможете оперативно реагировать на обратную связь от стейкхолдеров, клиентов и рынка.

Короткие спринты укрепляют доверие стейкхолдеров

Поскольку команды чаще демонстрируют результаты, ход работ становится прозрачнее, поэтому для стейкхолдеров его продвижение налицо, а это укрепляет доверие между ними и командой.

Короткие спринты способствуют скорости разработки посредством циклов «изучайте и приспосабливайтесь»

Чем выше частота итераций, тем больше у команды возможностей поразмышлять над полученным опытом, сделать выводы и применить их результаты к практическим методам в работе. Для этой области подходит то же самое обоснование, что и для оперативности работы с клиентами: что лучше — позволить своим командам изучать и приспосабливаться 26 раз в год или только четыре? Короткие спринты помогают вашей команде совершенствоваться быстрее.

Короткие спринты помогают сократить время на эксперименты

В контексте «Запутанные» по Cynefin проблемы нужно сначала исследовать, прежде чем получится понять весь объем работ. Такие исследования нужно характеризовать так: «Чтобы получить ответ на тот

или иной вопрос, сделайте наименьшее необходимое количество работы». К сожалению, закон Паркинсона гласит, что работа занимает весь доступный объем времени. А до тех пор, пока у команды не разовьется железная дисциплина, решение вопроса, если на него выделен месяц, и займет ровно месяц. Но если есть всего две недели, то чаще всего и решение вопроса занимает две недели.

Короткие спринты позволяют вовремя обнаружить издержки и риски

Короткие спринты дают возможность чаще отслеживать состояние хода работ. По истечении лишь нескольких спринтов при работе над новой задачей выяснится «скорость» команды или темп хода работ. Возможность наблюдения за продвижением работы облегчает прогноз сроков выпуска продукта в релиз. Если работа занимает больше времени, чем изначально планировалось, это станет предельно ясно всего через несколько недель. Небольшая длительность спринтов — мощный инструмент для осознания положения дел. В главе 20 «Еще более эффективная предсказуемость» рассказано об этом более подробно.

Короткие спринты способствуют подотчетности команды

Если команда несет ответственность по доставке рабочего функционала каждые две недели, у нее нет возможности долго работать «в темную». Она показывает результаты своей работы на встречах для обзора спринта, а также отчитывается перед стейкхолдерами каждые две недели. Еще чаще плоды труда видит владелец продукта.

Владелец продукта принимает или отклоняет работу, ход работы хорошо просматривается. Таким образом, команды лучше отчитываются за свою работу.

Короткие спринты способствуют подотчетности

На протяжении поколений команды разработчиков страдали от «звезд», которые запирались на несколько месяцев в темную комнату, и было совершенно неясно, продвигается ли работа. В случае со Скрамом такой проблемы больше нет. Каждый работает в поддержку целей команды в спринте, к тому же существует необходимость каждый день приходить на стендапы и рассказывать о том, какая работа вчера была выполнена, — запереться больше не получится. Либо разработчик начинает сотрудничать, что решает проблему одним способом, либо не выносит условий и уходит из команды, что все равно решает проблему, хотя и другим способом. По своему опыту могу сказать, что любой из исходов благоприятнее, чем в случае, когда кто-то работает без каких-либо отчетов

недели или месяцы, а в итоге обнаруживается, что ничего толком не сделано.

Короткие спринты способствуют автоматизации

Поскольку команды часто выполняют сверку, спринты небольшой длительности способствуют автоматизации задач, которые бы в ином случае были бы однообразны и затратны по времени. Автоматизация распространена в том числе при сборке, интеграции, тестировании и статическом анализе кода.

Короткие спринты чаще доставляют чувство удовлетворенности

Команда, которая доставляет работающее программное обеспечение каждые две недели, с завидной частотой ощущает удовлетворенность своей работой, а также чаще располагает возможностью отметить свои достижения. Это вносит свой вклад в чувство профессионализма, которое способствует мотивации.

Короткие спринты. Резюме

В целом пользу коротких спринтов можно кратко характеризовать так: «Скорость доставки во всех отношениях помогает справиться с объемом работ». Доставка функционала небольшими партиями и короткие каденции приносят огромное количество преимуществ по сравнению с доставкой функционала большими партиями и длинными каденциями.

Планируйте в зависимости от скорости выполнения задач

Единицы сложности историй — средство измерения размера и сложности той или иной задачи. Скорость — это показатель продвижения работы, основанный на частоте выполнения задач и измеряемый в единицах сложности историй. Планирование на основе скорости — это планирование работы и отслеживание ее хода на основании единиц сложности истории и скорости работы.

Планирование в зависимости от скорости выполнения задач и отслеживание не входят в учебник по Скраму, но я исходя из своего опыта, думаю, что не мешало бы их туда включить. Дальше расскажу о том, как применяются единицы сложности историй и оценка скорости.

Оценка размера бэклога продукта

Оценка единиц сложности применяется для определения размера бэклога продукта. Размеры задач в бэклоге продукта оценивают, как правило, посредством единиц сложности историй, а потом единицы сложности историй складывают, чтобы рассчитать общий размер бэклога продукта.

Это нужно сделать в начале цикла релиза и по мере добавления или удаления работы из бэклога. Это делается в той степени, которая необходима команде для обеспечения предсказуемости. О ней — в главе 20 «Еще более эффективная предсказуемость».

Расчет скорости

Объем работ, выполненных командой в каждом спринте, исчисляется единицами сложности историй. Количество единиц сложности историй, которого удается достичь в каждом спринте, означает скорость команды. Скорость рассчитывают на основе результатов каждого спринта посредством вычисления среднего значения.

Планирование спринта

Команда осуществляет планирование объема работ на спринт, также измеряемый единицами сложности историй, на основе наблюдений за скоростью команды.

Если команда в среднем выполняла 20 единиц сложности историй за спринт, а в следующем спринте ставит себе цель выполнить 40 единиц, то ей нужно сократить свои планы. Если один из членов команды собирается в отпуск или несколько членов команды посещают мероприятия по подготовке, команде стоит запланировать меньше единиц сложности историй на спринт, чем она выполняла в среднем. Если команде удалось выполнить 20 единиц сложности историй благодаря бессонным ночам и работе в выходные, также следует понизить планку. Если команда стабильно и без труда достигает целей спринта, попробуйте запланировать больше единиц сложности историй, чем выполняли в среднем. В любом случае, команда ведет планирование, исходя из своей реальной скорости.

Отслеживание релиза

На основе средней скорости можно рассчитать или спрогнозировать количество времени, необходимое на выполнение задач в бэклоге продукта. Если продукт бэклога насчитывает 200 единиц сложности историй, а средняя скорость команды составляет 20 единиц сложности историй за спринт, то команде потребуется 10 спринтов для выполнения всех задач из бэклога. Об особенностях того, как это устроено, я расскажу больше в главе 20 «Еще более эффективная предсказуемость».

Учет влияния изменений процесса, состава команды и других изменений

На основе скорости можно проследить влияние изменений процесса, состава команды и других изменений. Особенности этого обсуждаются в главе 19 «Еще более эффективное совершенствование процесса».

Ключевой принцип: доставка продукта вертикальными срезами

Чтобы спринты были эффективны, команде нужно развить способность частой доставки небольших партий рабочего функционала. Подход к проектированию, способствующий этому, называется «использование вертикальных срезов», что означает внесение изменений в каждый архитектурный слой для получения инкремента или ценности.

Вертикальный срез представляет собой полную функциональность стека, например добавление поля в банковскую выписку или обеспечение подтверждения транзакции пользователя на секунду быстрее. Каждый из этих примеров обычно требует работы всего технологического стека, как это видно из рис. 9.4.

Вертикальные срезы, как правило, помогают стейкхолдерам, не знающим технических тонкостей, лучше понимать, наблюдать и оценивать ценность для бизнеса. Они дают команде возможность выпускать релизы быстрее и осознавать реальную ценность продукта для бизнеса, получать настоящую обратную связь от пользователей.

Команды, использующие горизонтальные срезы, рискуют погрузиться в дебри на несколько спринтов подряд, работая над историями, которые в какой-то мере отражают прогресс, но не приносят видимой ценности для бизнеса.

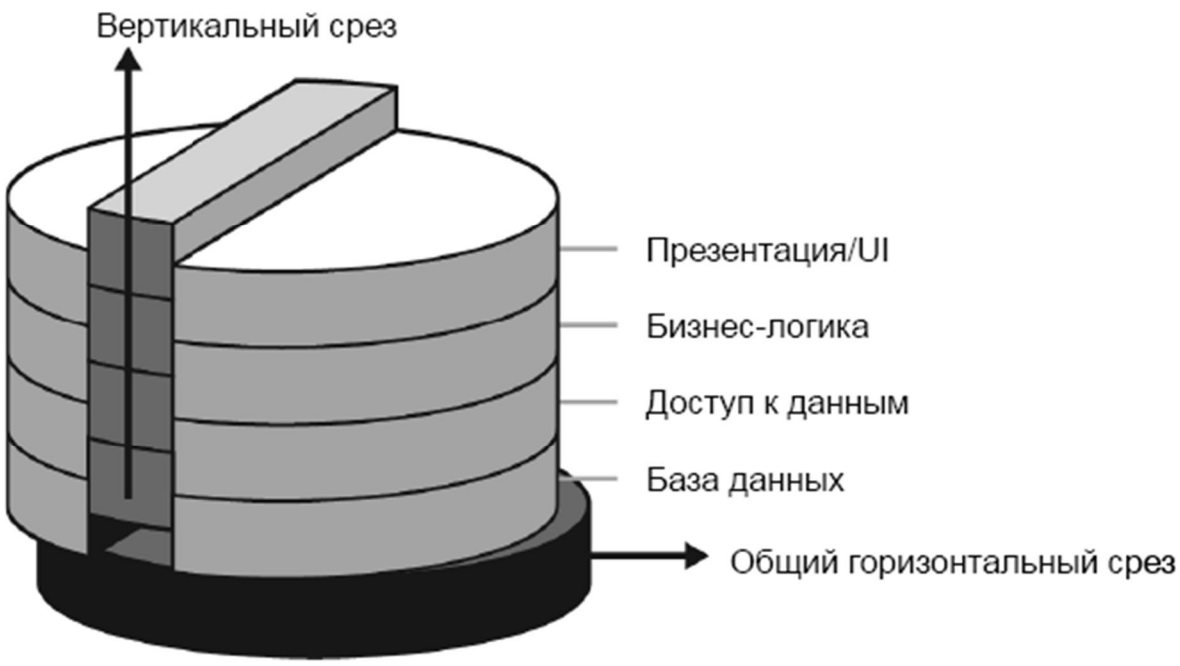


Рис. 9.4. Использование горизонтальных и вертикальных срезов. Использование вертикальных срезов включает работу во всех архитектурных слоях для получения инкремента функционала

Команды иногда не желают использовать вертикальные срезы, обычно по причине более низкой эффективности. Они будут утверждать, к примеру, что эффективнее выполнить больше работы в слое бизнес-логики, а потом уже перейти к слою интерфейса. Такой подход и называется использованием горизонтальных срезов.

В некоторых случаях с технической точки зрения, возможно, и эффективнее работать по горизонтальным слоям, но такая техническая

эффективность, как правило, приводит к оптимизации отдельных частей продукта, что не так важно, как получение ценного функционала. Вопреки утверждениям о том, что использование горизонтальных срезов ведет к увеличению эффективности, наша компания обнаружила, что при использовании горизонтальных срезов многие команды сталкиваются с необходимостью внесения значительных исправлений.

Вертикальные срезы обеспечивают более тесную обратную связь

Вертикальные срезы позволяют быстрее предоставить функционал бизнес-пользователям, что способствует быстрому получению обратной связи по поводу правильной работы функционала.

Поскольку вертикальные срезы требуют сквозной разработки, члены команды вынуждены работать совместно над проектированием и реализацией, что обеспечивает полезную техническую обратную связь для всей команды.

Использование вертикальных срезов также способствует сквозному тестированию, которое помогает обеспечить тесную обратную связь.

Вертикальные срезы позволяют получить бˆольшую ценность для бизнеса

Вертикальные срезы лучше понятны стейкхолдерам, не знающим технических тонкостей, а это приводит к улучшению качества решений, которые бизнес принимает в отношении приоритета и очередности реализации нового и пересмотренного функционала.

Поскольку вертикальные срезы позволяют получить инкремент функционала полностью, они тем самым чаще дают возможность представить рабочий функционал пользователям, что повышает ценность для бизнеса.

Использование горизонтальных срезов приводит к тому, что разработчики начинают воспринимать в качестве продукта архитектуру. А это может приводить к поощрению деятельности, которая совершенно не нужна для доставки функционала, и к прочим методам, которые приводят к снижению ценности.

Что нужно команде для использования вертикальных срезов

Доставка функционала вертикальными срезами бывает трудна. Это зависит от состава команды, ее способностей в бизнесе, разработке и тестировании, которые включают в себя навыки работы по всему технологическому стеку.

Командам также может понадобиться изменить свой образ мышления при проектировании и реализации, чтобы работать вертикальными срезами, который будет отличаться от работы по компонентам или

горизонтальным слоям. Некоторым командам для этого не хватает навыков проектирования, и чтобы с этим справиться, придется такие навыки развивать (и поддерживать развитие).

Наконец, командам нужно и получать работу вертикальными срезами. Владелец продукта и команда разработчиков должны найти такой подход к уточнению бэклога, чтобы в результате получались вертикальные срезы.

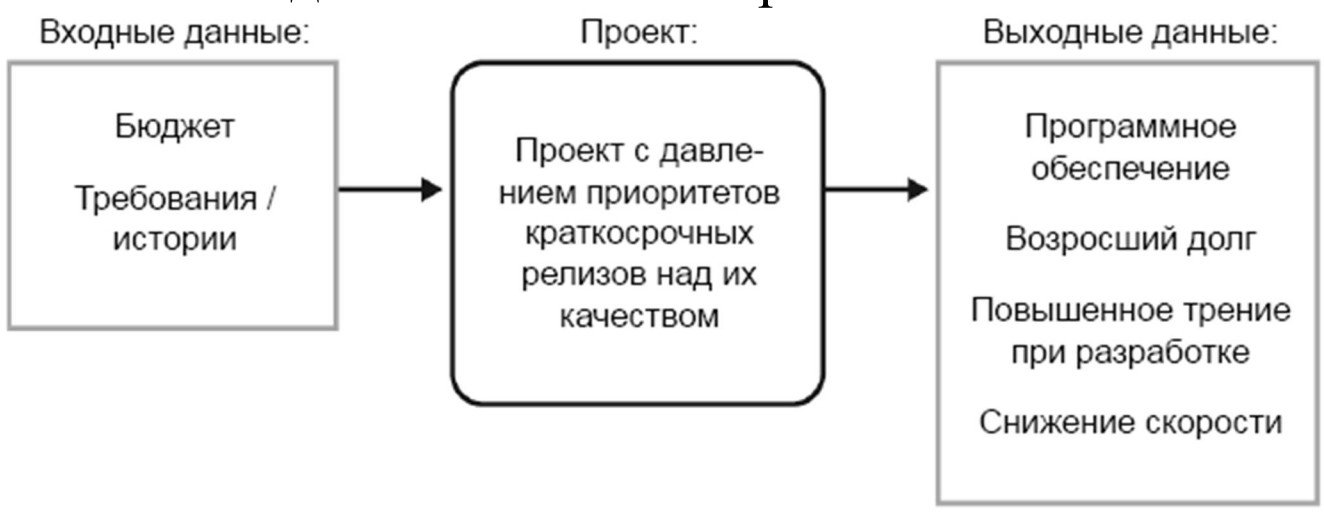
Ключевой принцип: управлЯЙТЕ техническим долгом

«Технический долг» означает накопление работы низкого качества в прошлом, которая замедляет темп работы в настоящем. Классический пример — это хрупкая кодовая база, в которой каждая попытка исправить ошибку порождает одну или несколько новых. Даже исправление простой ошибки ведет к затратам времени и исправлению дополнительных ошибок.

Технический долг может включать в себя низкокачественный код, низкокачественное проектирование, слабый набор тестов, трудный подход к проектированию, громоздкую среду сборки, медленные ручные процессы и другие способы, которыми команда жертвует долговременной производительностью в пользу получения краткосрочных достижений.

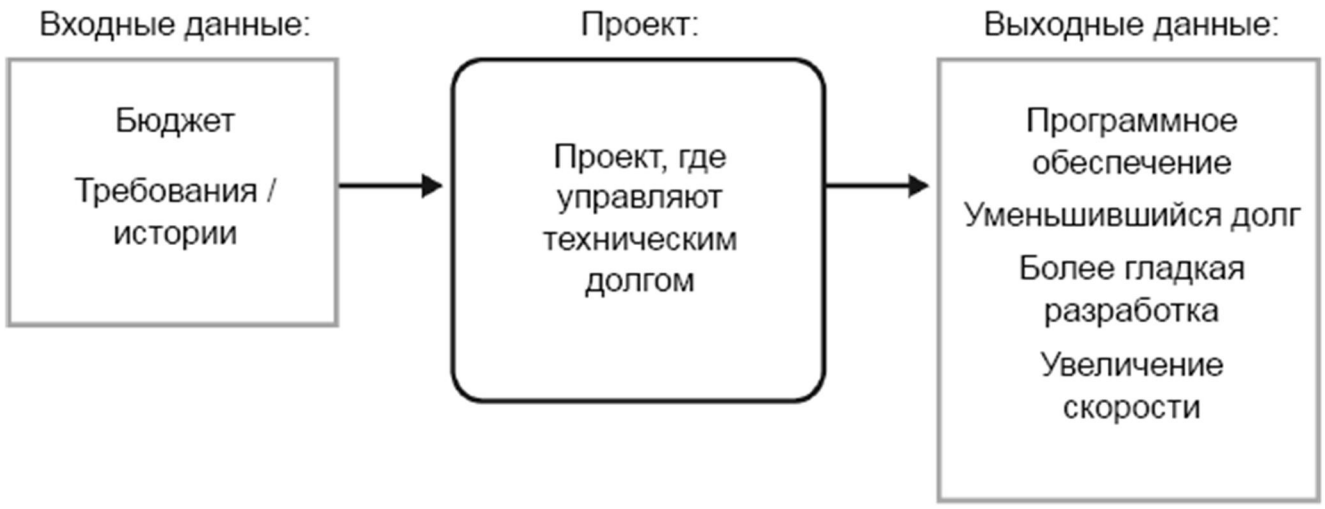
Последствия технического долга

Долг обычно накапливается в результате давления приоритета краткосрочных релизов в ущерб качеству. Целостное представление о вложенных ресурсах и полученной отдаче включает в себя учет влияния технического долга с течением времени.



У команд технических и бизнес-специалистов могут быть веские причины для накопления такого долга. Некоторые релизы достаточно чувствительны ко времени, чтобы гарантировать дополнительный объем работ впоследствии из-за желания быстрее выполнять работу в настоящем.

Тем не менее модель, по которой позволено со временем накапливать технический долг, без плана проработки такого долга в конечном итоге обеспечит снижение скорости команды. Команде нужно выработать план поддержания долга на регулируемом уровне, чтобы поддерживать или увеличить свою скорость.



Крухтен, Норд и Озкая разработали отличную для понимания схему того, как появляется технический долг, какую (вероятную) ценность для бизнеса он имеет и как, в конечном счете, становится больше обязательством, чем активом (рис. 9.5).

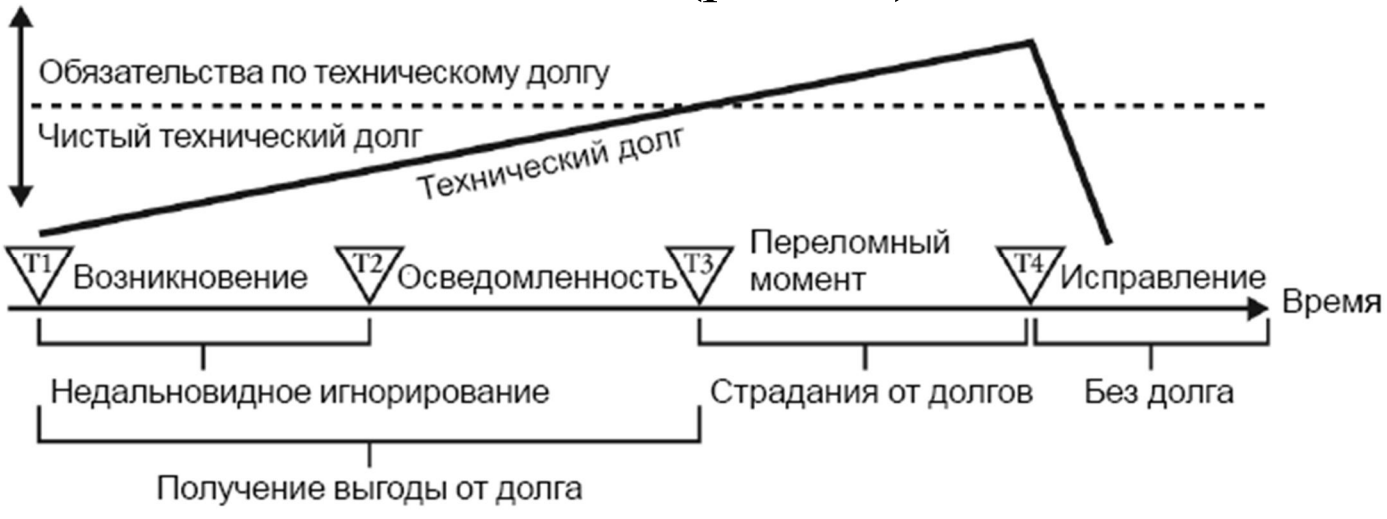


Рис. 9.5. Схема технического долга [Крухтен, 2019]

При работе с нуля команды могут в первую очередь избегать накопления технического долга. При выполнении уже начатой работы командам зачастую больше ничего не остается, как работать с уже накопленным техническим долгом. При выполнении любых видов работ, если команда плохо справляется с техническим долгом, скорость со временем снизится.

Погашение технического долга

У разных команд разный подход к погашению такого долга. Некоторые команды, чтобы погасить долг, распределяют его по долям на каждый цикл разработки (спринт или релиз). Другие команды добавляют долги в бэклог продукта или список недочетов и расставляют приоритеты для выполнения долгов и остальной работы. В любом случае, ключевая особенность заключается в том, чтобы технический долг регулировался открыто.

Виды долга и как с ним справиться

Не все технические долги одинаковы, и существуют их разнообразные классификации. Я нахожу полезными эти категории:

- *Намеренный долг (краткосрочный).* Долг, полученный из тактических или стратегических соображений, например выпуск чувствительного ко времени релиза в срок.
- *Намеренный долг (долгосрочный).* Долг, полученный из стратегических соображений, например по причине изначальной поддержки единственной платформы вместо поддержки кросс-платформенности.
- *Непреднамеренный долг (недобросовестность).* Долг, который по воле случая возникает из-за низкокачественных методов разработки. Такой вид долга замедляет работу как в настоящем, так и в будущем, поэтому его нужно избегать.
- *Непреднамеренный долг (добросовестность).* Долг, который возникает случайно из-за естественных ошибок при разработке ПО («Наш подход к проектированию не так хорошо сработал, как мы планировали» или «Новая версия платформы показала серьезные недочеты проектирования»).
- *Унаследованный долг.* Долг, унаследованный новой командой от старой кодовой базы.

Табл.9.1 показывает, какие подходы рекомендованы для реагирования на перечисленные виды долга.

Таблица 9.1. Виды технического долга и меры реагирования

Вид долга	Рекомендуемые меры
Намеренный (краткосрочный)	Допустите долг, если того требуют соображения бизнеса, как можно быстрее погасите его
Намеренный (долгосрочный)	Допустите долг при необходимости, определите условия погашения долга
Непреднамеренный (недобросовестность)	Применяйте практические методы высокого качества, чтобы в первую очередь избежать такого долга
Непреднамеренный (добросовестность)	Такой долг сам по себе неизбежен. Отследите влияние долга и погасите его тогда, когда это целесообразнее всего
Унаследованный	Создайте план постепенного погашения долга

Польза обсуждения технического долга

По моим наблюдениям, метафора «технический долг» оказалась полезной для облегчения обсуждений между техническими специалистами и специалистами по бизнесу. Специалисты по бизнесу, как правило, не знают, каковы издержки погашения технического долга, а технические специалисты, как правило, не знают интересы бизнеса. В некоторых случаях будет хорошим решением намеренно допустить образование технического долга, а в каких-то нет. Понятие долга облегчает обмен мнениями о технических соображениях и соображениях бизнеса, делая его более содержательным, что улучшает качество решений о том, когда и зачем принимать на себя долг и когда и как его гасить.

Выстройте структуру работы во избежание выгорания

Пуристский взгляд на Agile предполагает одинаковую длительность спринтов (известную как «общая каденция»). Если команда хорошо переносит общую каденцию, нет смысла вносить изменения. Общие

каденции позволяют упростить расчет скорости и других факторов при планировании спринта.

Распространенная жалоба при внедрении Скрама заключается в том, что бесконечная последовательность спринтов приводит к усталости и ощущению бега в колесе. При последовательной разработке присутствуют естественные провалы в работе, особенно между дисциплинами, а также достижение баланса в периоды высокой интенсивности.

Постоянные спринты не оставляют времени на отдых, если каждый спринт действительно проходит в темпе спринта.

Одно из противоядий от усталости после спринтов — изменение по необходимости длительности спринтов. Системный способ его обеспечить — использовать шаблон 6 × 2 × 1, шесть спринтов по две недели плюс один спринт длительностью одну неделю, в сумме 13 недель, которые как раз составляют один квартал.

Альтернативой этому послужит применение сокращенных спринтов после крупных релизов, в праздничные дни и в другое время, когда скорость команды все равно не будет сохранять стабильность. Во время недельного спринта команда может работать над инфраструктурой или инструментами, посещать подготовительные или командные мероприятия, хакатоны, прорабатывать технический долг, работать над улучшениями, которые слишком велики для того, чтобы успеть их выполнить за спринт, или делать что-то еще.

Различная длина каденции спринта соответствует понятию устойчивого темпа, применяемого в Agile. Многие исследования по Agile приравнивают устойчивые темпы к отсутствию свободных вечеров и выходных. Но я думаю, что это досадное упрощение, которое игнорирует различия в предпочтениях условий труда разных людей. В качестве устойчивого темпа некоторыми предлагается 40-часовая рабочая неделя, но для других это прямая дорога скуке. Лично я большую часть лучшей своей работы проделал во взрывном режиме — 55 часов в течение двух недель и затем 30 часов в течение последующих двух недель. В среднем может получиться около 40 рабочих часов в неделю, но у разных команд рабочие недели не всегда составляют 40 часов. Понимание устойчивого темпа у всех разное.

Прочие соображения

Внепроектная разработка ПО

Не вся разработка ПО организована в проекты, даже с учетом множества определений, приведенных в начале главы. Иногда возникают ситуации, в которых обычно работает один человек, например это распространено в

обработке обращений в техническую поддержку, решении проблем выпуска в производство, создании патчей и тому подобном.

Такие работы, конечно же, относятся к разработке ПО и также поддаются практическим методам Agile. Их можно проводить более эффективно, качественно, методично благодаря внедрению таких практических Agile-методов, как бережливое производство и Kanban. Но по моему опыту, компании обычно испытывают гораздо меньше проблем с работой такого характера, чем с разработкой ПО в масштабах проекта, поэтому основное внимание в этой книге уделяется работе над проектами, а не над ситуативной работой.

Рекомендации

Изучайте

- Просмотрите историю итогов проектов. Соответствует ли опыт вашей компании общему мнению о том, что небольшие проекты чаще выполняются более успешно, чем крупные?
- Просмотрите портфолио своих проектов. Какие из ваших крупных проектов можно поделить на несколько небольших?
- Просмотрите каденции, по которым работают ваши команды. Сколько времени длятся спринты? Меньше чем три недели?
- Узнайте, доставляют ли ваши команды функционал вертикальными срезами.
- Узнайте, осуществляют ли ваши команды планирование на основании скорости.
- Опросите свои команды о техническом долге. Как они воспринимают бремя долга и получится ли у них погасить его?

Приспосабливайтесь

- Поощряйте ваши команды учитывать скорость работы при постановке целей спринта.
- Разработайте план, по которому команда сможет доставлять работу вертикальными срезами, в том числе при проектировании и при уточнении бэклога владельцем продукта.
- Поощряйте создание командами планов по регулированию технического долга.

Дополнительные ресурсы

[Фред Брукс, 1975]. Мифический человеко-месяц. Несмотря на то что книга не нова, она содержит классическое обсуждение трудностей, возникающих на пути к успеху при выполнении крупных проектов.

[Стив Макконнелл, 2019]. Understanding Software Projects Lecture Series. Серия Construx OnDemand. Сетевой ресурс (2019 год). https://ondemand.construx.com. В этих лекциях подробно обсуждается взаимосвязь размера проекта и динамики разработки ПО.

[Кеннет Рубин, 2012]. Essential Scrum: A Practical Guide to the Most Popular Agile Process. В этом всеобъемлющем руководстве по Скраму дано

объяснение, как на основе единиц сложности и скорости команды планировать спринты и релизы.

[Филипп Крухтен и др., 2019]. Managing Technical Debt. Это во всех отношениях полноценное и продуманное рассуждение на тему технического долга.

ГЛАВА 10. Еще более эффективное выполнение крупных проектов

Натуралист Стивен Джей Гулд рассказывает историю, в которой две маленьких девочки разговаривают на детской площадке [Гулд, 1977]. Одна девочка говорит: «Что, если паук был бы большим, как слон? Было бы страшно, да?» А другая девочка отвечает: «Нет. Если бы паук был большой, как слон, то он бы и выглядел, как слон. Что тут непонятного?»

Дальше Гулд объясняет, что вторая девочка права, потому что внешний вид организма напрямую зависит от его размеров. Паук может парить в воздухе без опаски для себя, потому что сопротивление воздуха для него сильнее, чем притяжение. Но слон для этого слишком тяжел. Для него притяжение гораздо сильнее, чем сопротивление воздуха. По мере роста паук может сбросить экзоскелет и образовать новый, потому что старый уже слишком мал, но слон слишком большой, чтобы линять и ждать, пока вырастет новый экзоскелет, поэтому ему необходим внутренний скелет. Гулд приходит к выводу, что если бы паук был таким же крупным, как слон, то он бы и напоминал больше слона в силу анатомии при таком размере.

При обсуждении проектов по разработке ПО встает аналогичный вопрос: «Что, если проект, выполняемый по Agile, был бы крупным? Было бы страшно, да?» Что ж, страшно, может, и не было бы, но здесь логика такая же, как и в случае с пауком и слоном.

Чем на самом деле различается Agile в крупных проектах?

На самом деле вопрос о том, как добиться эффективности при выполнении по Agile крупных проектов, не совсем правильно поставлен. Организации испытывают трудности со всеми видами крупных проектов с того времени, как появилась разработка ПО [Брукс, 1975]. Трудности также бывали и с небольшими проектами.

Практические Agile-методы, и Скрам в частности, дали возможность чаще успешно справляться с небольшими проектами, и основное внимание перешло на крупные проекты, с которыми до сих пор много проблем.

Принципы Agile в крупных проектах

Разные компании совершенно по-разному могут трактовать понятие «крупный». Нам встречались организации, в которых любой проект, где требовалось больше одной типичной по размеру команды, считался крупным, а были и такие организации, для которых проект, вовлекавший менее сотни человек, считался средним или небольшим. «Крупный» — растяжимое понятие. Соображения, приведенные в этой главе, уместны тогда, когда вовлечено две или больше команд. Некоторые принципы Agile подходят к крупным проектам, а некоторые нужно доработать. В табл. 10.1 показано, как принципы Agile работают в крупных проектах.

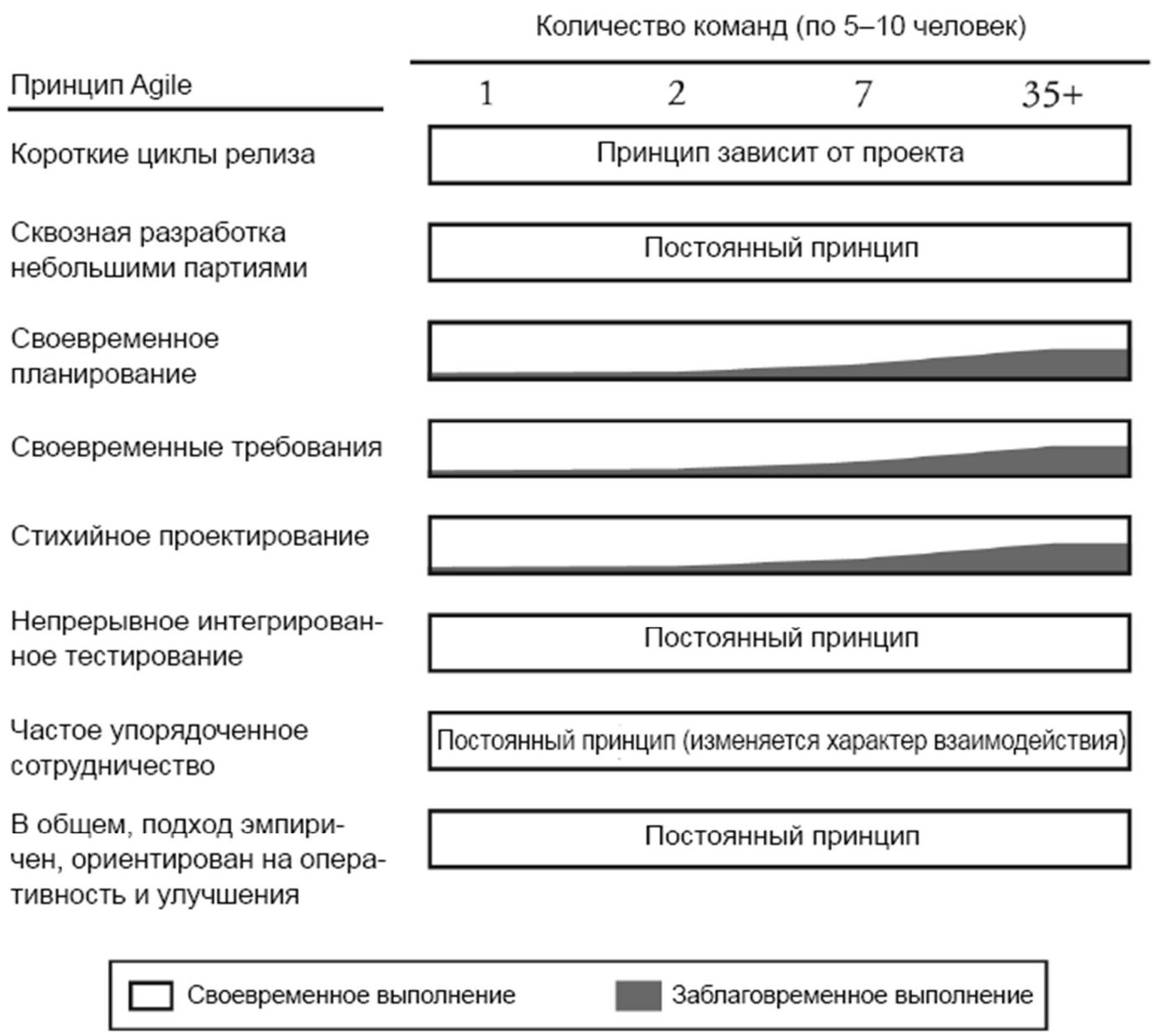
Таблица 10.1. Принципы Agile в крупных проектах

Принцип Agile	Применительно к крупному проекту
Короткие циклы релиза	Лучше всего организовать команды соответствующего размера с коротким циклом релиза
Сквозная разработка ведется маленькими партиями	Без изменений, команды могут по-прежнему вести сквозную разработку мелкими партиями, хотя требуется более высокий уровень координации
Заблаговременное планирование на общем уровне и своевременное планирование подробностей	Долю заблаговременного планирования нужно увеличить
Заблаговременные требования на общем уровне и своевременные требования в подробностях	В крупных проектах нужна бˆольшая координация требований, что означает необходимость большего количества времени от начала их уточнения до завершения реализации
Стихийное проектирование	Издержки ошибок и перепроектирования растут с увеличением размера проекта, это тот принцип Agile, который в первую очередь требует доработки при использовании в крупном проекте
Непрерывное тестирование, интегрированное в разработку	Эти принципы потрясающе работают вне зависимости от размера проекта. В крупных проектах большее внимание уделяется интеграционному и системному тестированию
Частое упорядоченное сотрудничество	Этот принцип обладает еще большей важностью в крупных проектах, изменяются способы взаимодействия в некоторых ситуациях
В общем, подход эмпиричен, ориентирован на оперативность и улучшения	Этот принцип работает хорошо как в крупных, так и в небольших проектах

Принцип Agile, касающийся ведения сквозной разработки малыми партиями, способствует эффективности работы над крупными проектами, так же как и непрерывное тестирование, частое упорядоченное сотрудничество и петля Бойда.

Крупные проекты требуют большей доли заблаговременного планирования, требований и проектирования. Не все нужно делать заранее, как при последовательной разработке, но явно больше, чем в обычном Agile. Это важно при планировании спринтов, обзорах спринтов, структурировании бэклога продукта, уточнении бэклога, планировании релиза и составлении диаграммы сгорания задач до релиза. В крупных проектах польза непрерывного тестирования не меньшая, чем в небольших проектах, но подход к тестированию нужно доработать — нужно проводить больше интеграционных и системных тестов.

Приведем иллюстрацию того, как принципы Agile изменяются по мере увеличения размеров проекта:



В следующих разделах описаны особенности адаптаций, необходимых для успешного выполнения крупных проектов по Agile.

Закон Брукса

Один из взглядов на применение еще более эффективного Agile в крупных проектах изложен Фредом Бруксом в его книге «Мифический человеко-месяц» [5]. В процессе обсуждения закона Брукса — идеи о том, что привлечение новых специалистов на поздних этапах проекта может его затормозить, — Брукс замечает, что если работу можно полностью распределить, то закон не всегда справедлив.

Это имеет непосредственное отношение к обсуждению крупных проектов, так как лучше всего крупный проект *полностью разбить на несколько отдельных* проектов поменьше. Если вам удастся так сделать, вы выиграете во многих отношениях. Увеличится удельная производительность на одного специалиста, сократится количество ошибок, как уже было сказано в главе 9 «Еще более эффективное выполнение проектов». Вы также дадите возможность сделать больший акцент на методах Agile, чем на методах последовательной разработки.

Но, как отмечает Брукс, задача разбить крупный проект по нескольким мелким состоит в достижении цели полного распределения работы. Полностью распределить работу трудно, а если работа *распределена только в общем*, это означает, что по-прежнему требуется координация разных команд проекта, небольшие проекты начинают все больше напоминать крупные. В этом случае все старания пройдут даром.

Закон Конвея

У вас не получится понять суть крупных проектов и получить максимальную отдачу от применения Agile без понимания закона Конвея. Как я писал в главе 7 «Еще более эффективные географически распределенные команды», закон Конвея утверждает, что структура продукта копирует структуру самой организации, которая разработала продукт.

Если техническое решение основано на большой монолитной архитектуре, команде проекта придется стараться изо всех сил, чтобы создать на этой основе что-то не большое и не монолитное.

Из сочетания идей Конвея и Брукса следует, что при работе по Agile в крупных проектах лучшей для большой системы будет та архитектура, которая будет *полностью*поддерживать распределение работы команд, работающих над ней. Этот идеал при создании одних продуктов достижим проще, чем при создании других. Унаследованные продукты, в частности, обычно требуют подхода «ползи, иди, беги».

Ключевой принцип: поддержка крупных Agile-проектов с помощью архитектуры

Чтобы архитектура продукта полностью поддерживала распределение работы, нужно потрудиться. Архитектуру некоторых старых продуктов можно доработать так, чтобы степень взаимосвязи была низкой, однако при построении архитектуры новых продуктов нужно заблаговременно распределить работу по нескольким небольшим командам.

Некоторые команды при работе по Agile будут против идеи большего заблаговременного проектирования, считая, что это уже «не Agile». Но, как подразумевал Стивен Джей Гулд, если выбирать подход, основные принципы которого ориентированы на проекты небольшого размера, приходится чем-то жертвовать, чтобы этот подход сработал с крупным проектом. Не стоит ждать успеха в ведении крупных проектов, не внося никаких изменений.

Если полностью рассматривать закон Конвея, то единственное, что действительно нужно изменить, — это принцип стихийного проектирования, а для этого нужно соответствующее планирование. Заблаговременное построение архитектуры, а также курс на полное распределение работы помогут организовать мелкие команды, поэтому остальные принципы Agile можно оставить неизменными. Стихийное проектирование также можно оставить в тех областях, где работа тщательно распределена между небольшими командами.

Не случайно вышло так, что распространение Agile и акцента на небольших командах совпало с возникновением микросервисной архитектуры.

Цель микросервисной архитектуры — создание приложений, структура которых будет представлять собой слабо связанные сервисы. Точно так же цель организации крупного Agile-проекта состоит в том, чтобы организовать людей в структуру слабо связанных небольших команд.

Организация, которая успешно справляется с построением архитектуры крупной системы, обеспечивающей поддержку полного распределения работы, не будет чувствовать, что работает над крупным проектом. Она будет рассматривать проект как коллектив из небольших команд, работающих независимо, которые объединяет только одно — они вносят вклад в общую кодовую базу.

Недостаток в построении архитектуры ведет к тому, что специалисты называют «эффектом снежинок», — все разрабатываемые функции, словно неповторимые снежинки, проектируются по-разному. Это накладывает значительные издержки на разработку, поскольку членам команд приходится подробно изучать каждую функцию, для того чтобы эффективно работать в каждой области кода. И чем больше становится проект, тем крупнее такие издержки. Если «снежинок» накапливается слишком много, в конце концов, вы рискуете попасть под лавину!

Конкретные предложения по архитектуре

Эта книга не учебник по архитектуре, но в этом разделе содержится поверхностное описание подходов к построению архитектуры, которые годятся для небольших команд при работе с крупными проектами. Раздел предназначен для технических специалистов, поэтому можете спокойно пропустить его, если техническая сторона вопроса вас не интересует.

Основы: слабая связанность, модульность

Старайтесь строить архитектуру со слабой связанностью (модульную и многослойную), с хорошо читаемым и несложным кодом.

Архитектура не обязательно должна представлять собой превосходно проработанный код для микросервисов. Она просто должна быть достаточно гибкой для соответствия требованиям бизнеса.

Иногда говорят, что заветная цель заключается в том, чтобы разбить систему, скажем, на 50 микросервисов. Они могут быть разделены на множество модулей, которые запускаются в собственных контейнерах со своими базами данных. У каждого из этих модулей может быть своя собственная версия и аутентификация API. Каждый из них можно запускать в производство и масштабировать независимо, что приближает к цели полностью распределить работу между 50 командами разработчиков.

Конечно, такая ситуация из разряда фантастики, но иногда это и вправду работает! Но если некоторые компоненты вашей системы обращаются к прочим частям системы, которые, в свою очередь, обращаются ко множеству других частей этой системы (так называемый «высокий коэффициент разветвления (fan-out)»), вы можете в конечном итоге столкнуться со значительными издержками на обработку вызовов и на взаимодействие между командами, работающими над различными микросервисами. Вероятно, как для структуры ПО, так и для структуры команд будет лучше объединить некоторые сервисы системы в более крупные.

Правильность решений зависит от сочетания взглядов технических специалистов на проектирование и взглядов лидеров на организацию команды.

Избегайте монолитных баз данных

Избегание одной большой базы данных позволяет распределить работу между командами. Слабо интегрированные базы данных могут поддерживать слабую связанность и высокую модульность внутри команд. Но в зависимости от характера связей между частями системы можно также и усложнить взаимодействие между ними, а это ведет к значительным издержкам, опозданиям и шансам допустить ошибку. Чтобы узнать, в какой степени нужно проводить декомпозицию системы, чтобы команды были слабо связаны и при этом поддерживалось высокое качество технических решений, необходимо сочетание оценки технических специалистов и оценки руководства команд.

Используйте очереди

Разделение или сдвиг времени с помощью очередей может также помочь в создании слабой связанности между командами. Если говорить абстрактно, нужно ставить задачи, относящиеся к другой части системы, в очередь для последующей обработки. «Последующей» может означать и доли секунды.

Ключевая концепция в этом разделении заключается в том, что система не просто лишь выполняет большую часть своего кода в непосредственном жестком цикле запроса-ответа. Использование очередей позволяет поддерживать низкий уровень связанности между ключевыми частями функционала системы, который позволяет обеспечить низкую связанность архитектуры и команды разработчиков (другой пример закона Конвея).

Полезно бывает подумать о ключевых «швах» системной архитектуры. Шов представляет собой такую границу, внутри которой происходит большое количество взаимодействия, но за ее пределами этого взаимодействия немного. Для того чтобы обеспечить слабую связанность, соединяйте элементы архитектуры по швам, используя очереди.

Как в примере с микросервисами, можно зайти слишком далеко — 50 процессов, управляющих 50 очередями заданий с зависимостями, могут вызвать вереницу проблем связанности, которые могут обернуться проблемами худшими, чем изначальная проблема.

Применяйте контрактное программирование

Контрактное программирование — это подход в проектировании, в котором особое внимание уделяется интерфейсам [Мейер, 1992]. Считается, что у каждого интерфейса есть «предусловия» и «постусловия». Предусловия — это обещания, которые пользователь компонента предоставляет компоненту относительно условий, которые будут верны до того, как компонент будет использован. Постусловия — это обещания, которые компонент передает обратно остальной части системы относительно условий, которые будут верны к окончанию выполнения работы компонентом.

Принимая во внимание закон Конвея, вы можете применять контрактное программирование, для того чтобы устранить влияние технических зависимостей на рабочий процесс.

«Контракт» будет регулировать взаимодействие между частями программной системы, а также неявно устанавливать ожидания для взаимодействий между людьми.

Сдвиг видов сотрудничества в крупных проектах

Многие рабочие практические методы основываются на эффективности личного взаимодействия. Много информации существует только как часть устной традиции команды. Например, составители требований в Agile открыто говорят о том, что большая часть любых требований заключается в обсуждении требований. Обнаружилось, что это отлично работает в небольших проектах.

Крупные проекты привлекают больше специалистов, они больше распределены географически (в том числе в разных зданиях одного научного городка), длятся дольше, со временем к проекту присоединяются новые люди, а другие, поработавшие в проекте некоторое время, со временем его покидают.

Чтобы крупные проекты по Agile были успешными, не стоит ожидать, что все знания могут быть переданы устно. Больше работы требуется проводить заранее, и бˆольшую часть этой работы нужно документировать так, чтобы людям, отсутствующим при самих разговорах, было все понятно.

Трудности координации в крупных проектах

В общем, подходы к масштабированию проектов разработки ПО, не только проектов, выполняемых по Agile, страдают от неправильной диагностики того вида координации, который необходим по мере увеличения масштабов проектов. Чем больше становится ваш проект, тем выше будет потребность в дисциплинах, не связанных с кодом, — обработке требований, построении архитектуры, управлении конфигурацией, тестировании и контроле качества, управлении проектами и ведении процесса. Ключевой вопрос состоит в том, нужно ли в какой-либо из этих областей более быстрое масштабирование или большая координация взаимодействий между командами, чем в остальных.

По опыту можно сказать, что больше всего проблем возникает от требований. По моему опыту, наиболее часто проблемы координации больших проектов встречаются в следующем порядке:

- Требования (наиболее часто).
- Архитектура (системы с интенсивным проектированием).
- Управление конфигурацией/версиями.
- Тестирование и контроль качества.
- Управление проектами.
- Процесс.

Когда вы рассматриваете подходы к крупным проектам, можете использовать этот список, чтобы примерно знать, в каких областях в первую очередь могут возникнуть трудности. Вы также можете провести обзор всех крупных проектов своей компании, изучить наиболее распространенные источники трудностей и составить план координации на их основе.

Система показателей в Agile в крупном проекте

Мы пришли к выводу, что полезно оценивать эффективность проекта по основным проблемным областям, возникающим при выполнении крупных проектов по Agile. На рис. 10.1 показан пример звездообразной диаграммы крупного проекта. Для этой диаграммы приводится такая же расшифровка, как для системы показателей в Скраме.

0 Не применяется
2 Применяется редко и неэффективно
4 Применяется эпизодически с переменной эффективностью
7 Применяется постоянно и эффективно
10 Оптимизация

Рис. 10.1. Инструмент диагностики, показывающий эффективность крупного проекта в соответствии с ключевыми факторами успешного ведения крупного проекта

Жирная линия означает показатели, которые в среднем встречались моей компании на практике. Пунктирная линия означает показатели здорового крупного проекта. Чтобы иметь достаточные шансы на успех, крупный проект должен набрать 7 пунктов или больше.

Приведем больше подробностей по категориям эффективности:

- *Требования*. Практические методы обработки требований для нескольких групп, в том числе управление продукцией, бэклог продукта, уточнение бэклога, демонстрации системы и обзоры спринтов для нескольких команд.
- *Архитектура*. Масштабирование практических методов проектирования под размер проекта, архитектурное русло (*architectural runway*) или эквивалент.
- *Качество*. Практические методы, рассчитанные на несколько команд, в том числе системные ретроспективы или встречи по изучению и приспособлению, показатели уровня продукта и критерии готовности продукта.
- *Тестирование*. Инфраструктура, поддерживающая автоматическое тестирование в нескольких командах, тестирование интеграции, сквозное тестирование системы, тестирование производительности, безопасности и другое специализированное тестирование.
- *Управление проектами*. Управление зависимостями, планирование для нескольких команд или программного инкремента, Scrum-of-scrums, синхронизация решений владельца продукта, отслеживание уровня состояния продукта/диаграмма сгорания задач.
- *Управление конфигурациями/Развертывание*. Контроль версий кода и инфраструктуры, DevOps, конвейеры разработки, управление релизами.

Исходя из опыта отрасли в целом, средний крупный проект, который мы рассмотрели, эффективен гораздо меньше, чем средний небольшой проект.

Начните со Скрама

Наставление главы 4 «начать со Скрама» еще более актуально при работе с крупными проектами. Если у вас не получается успешно вести небольшие проекты, вы столкнетесь с еще большими проблемами в крупных проектах. Убедитесь, что ваши небольшие проекты, как правило, успешны, и отталкивайтесь от них. Как писали Барри Боэм и Ричард Тернер в своей работе Balancing Agility and Discipline, увеличение небольшого процесса обычно получается лучше, чем уменьшение крупного [Боэм, 2004].

Прочие соображения

Scrum-of-scrums

Scrum-of-scrums (Скрам Скрамов или Схватка Схваток) — это подход, направленный на расширение Скрама, при работе нескольких команд. В таких проектах встречи, посвященные Scrum-of-scrums, проходят один раз в неделю или чаще. Каждая команда отправляет своего представителя на встречи, похожие на дейли-скрам.

Хотя Scrum-of-scrums кажется логичным способом распределить большой объем работ по нескольким скрам-командам, на нашей памяти этот подход редко срабатывал. Я считаю, что одна из причин — это то, что в качестве представителей команды, как правило, на координационные встречи посылали скрам-мастеров. Таким образом, команда подразумевает, что основные трудности возникают в области процесса и в основе рабочего потока, но по опыту могу сказать, что больше всего трудностей возникает из-за требований. Гораздо полезнее отправить на встречу владельца продукта, чем скрам-мастера.

Масштабированный фреймворк Agile (SAFe)

Масштабированный фреймворк Agile (Scaled Agile Framework, SAFe) — это сложный набор методов для масштабирования Agile на крупных предприятиях. Подход SAFe на сегодня наиболее часто встречается при внедрении Agile в крупных проектах среди компаний, с которыми нам доводилось работать. SAFe хорошо продуман, он постоянно развивается и совершенствуется, в нем есть по-настоящему полезные составляющие. Несмотря на сказанное, только в нескольких компаниях, с которыми мы работали, были довольны своей реализацией SAFe, при этом реализации были в высокой степени модифицированы.

Когда мы работали с софтверными компаниями, то обнаружили, что все небольшие компании считают себя уникальными, хотя не являются такими на самом деле. У них одни и те же проблемы, которые можно исправить одним и тем же способом. А вот все крупные компании думают, что должна быть еще какая-то компания, в точности ее повторяющая, но такого не бывает. За некоторое время произошли рост, развитие, отладка

различных технических методов, методов ведения бизнеса, появилась собственная культура.

Скрам уместен в качестве шаблона для небольшого проекта. У SAFe нет возможности такого же универсального применения к крупным проектам, как у Скрама к небольшим. Его нужно хорошо приспособить, и часто до такой степени, что лучше уже задуматься о нем как об источнике инструментария, чем об интегрированном наборе методов. Если вы выбираете SAFe, рекомендуем начать с Essential SAFe (самая маленькая версия SAFe) и отталкиваться уже от этого.

Рекомендации

Изучайте

- Обсудите архитектуру вашей системы с вашими техническими руководителями с точки зрения закона Конвея. Каким образом организация взаимодействий между специалистами сказывается на технической организации и наоборот?
- Проведите обзор организации взаимодействий между специалистами в самых крупных из ваших проектов. В какой степени работа действительно распределена или монолитна? Насколько сложно организована сеть взаимодействий между специалистами? Как это отражается на архитектуре ПО?
- Пересмотрите принципы Agile, изложенные в табл. 10.1. Подумайте, нет ли какого-то более легкого альтернативного способа для организации специалистов, так чтобы соблюдались принципы из таблицы без большого количества заблаговременного проектирования.
- Проведите обзор тех трудностей, которые встречались в крупных проектах, и определите, в какой степени возникали координационные проблемы из-за требований, архитектуры, управления конфигурацией и версиями, тестирования и контроля качества, управления проектами или процесса.

Приспосабливайтесь

- Составьте план разработки архитектуры, чтобы организовать структуру слабо связанных команд.
- Пересмотрите свой подход к крупным проектам и учтите источники координационных проблем, которые вы обнаружили при изучении по совету выше.

Дополнительные ресурсы

[Стив Макконнелл, 2004]. Совершенный код (СПб.: Питер, 2007). В главе 27 описываются тенденции крупных проектов в сравнении с небольшими, особое внимание обращено на то, в каком соотношении изменяются действия на уровне проекта по мере роста его размера.

[Стив Макконнелл, 2019]. Understanding Software Projects Lecture Series. Серия Construx OnDemand. Сетевой ресурс (2019 год). https://ondemand.construx.com. Во многих из лекций этой серии уделяется внимание проблемам, связанным с размером проекта.

[Роберт С. Мартин, 2017]. Чистая архитектура. Искусство разработки ПО (СПб.: Питер, 2018). Это популярное руководство для построения архитектуры ПО, которое начинается с принципов проектирования и доходит до архитектуры.

[Лен Басс и др., 2012]. Software Architecture in Practice, 3rd Ed. Это комплексное рассуждение на тему архитектуры в стиле учебника.

[Барри Боэм и Ричард Тернер, 2004]. Balancing Agility and Discipline: A Guide for the Perplexed. Эта книга — ценный источник понимания особых случаев динамики зависимости размера проекта и применения Agile. Для опытного читателя. Для менее опытного читателя книга слишком устарела, увязнув в практических методах в районе 2004 года (главный метод — экстремальное программирование, не обсуждаются критерии готовности, допускаются долгие 30-дневные спринты, нет понятия уточнения бэклога и т.д.).

ГЛАВА 11. Еще более эффективное качество

«Если нет времени, чтобы сделать правильно, откуда тогда возьмется время на то, чтобы это переделать?» — такой вопрос из поколения в поколение задается в организациях, которые ставят на качество. Средства для того, чтобы «сделать правильно», неуклонно развиваются, и современный Agile внес несколько полезных практических методов.

Ключевой принцип: быстрее находите дефекты

Обычно мы об этом не думаем, но при работе над проектом постоянно появляются дефекты. Каждый час работы команды порождает некоторое количество дефектов. Таким образом, построение кривой накопления дефектов в проекте разработки практически то же самое, что построение кривой накопления усилий.

В отличие от допущения дефектов, их обнаружение и исправление не происходит в результате основной функциональной деятельности. Они происходят во время контроля качества.

Как показано в верхней части рис. 11.1, во многих проектах обнаружение и устранение дефектов происходит значительно позже их появления.

Это является проблемой, поскольку область между двумя кривыми представляет скрытые дефекты, которые уже образовались в программном продукте, но еще не были обнаружены и устранены. Каждый из этих дефектов влечет за собой дополнительную работу по устранению, которая редко укладывается в план. Каждый из таких дефектов создает работу, которая может стоить бюджету любых средств, сорвать график и проект в целом.

В эффективных проектах сужается промежуток между появлением ошибки и ее обнаружением и устранением, как показано в нижней части рисунка.

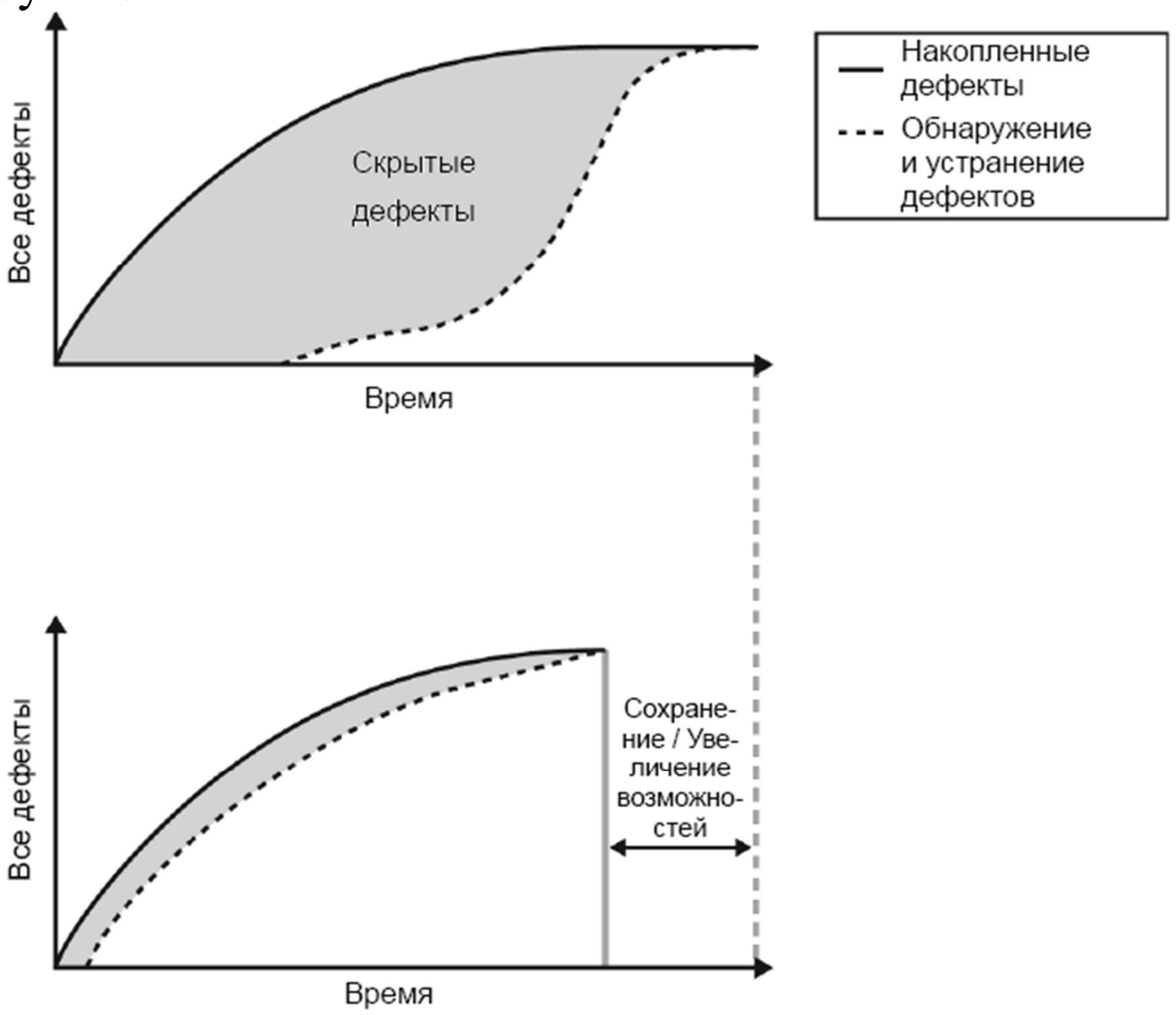


Рис. 11.1. Область между кривой накопления дефектов и кривой обнаружения и устранения ошибок обозначает скрытые дефекты

Проекты, в которых дефекты быстро исправляют, более эффективны. Как показано на рисунке, на выполнение таких проектов нужно меньше времени и меньше усилий. Ни в одном проекте не удается обнаружить 100% дефектов немедленно, но уменьшение количества скрытых недочетов — это полезная цель, даже несмотря на то, что она не может быть полностью достижимой. Если сравнить две схемы (рис. 11.1) с точки зрения готовности релизов, то проект на нижней диаграмме будет готов больше.

К практическим методам, помогающим обнаружить дефект как можно раньше, относится юнит-тестирование, парное программирование, статический анализ, код-ревью (если проводится своевременно) и непрерывное тестирование — они позволяют добиться эффективности на уровне мелочей. Акцент Agile на регулярном доведении ПО до уровня качества готового к выпуску релиза каждые 1–3 недели позволяет добиться эффективности на уровне крупных целей.

Ключевой принцип: создайте и применяйте критерии готовности

Ясные критерии готовности позволяют сокращать промежуток между появлением недочета и его обнаружением тем, что работа по контролю качества выполняется близко ко всей остальной работе.

Понятные критерии готовности включают критерии выполнения проектирования, написания кода, тестирования, написания документации и всей остальной работы, относящейся к реализации требований. Критерии выполнения лучше всего изложить в терминах, которые однозначно верны или ложны. На рис. 11.2 изображен пример критериев готовности.

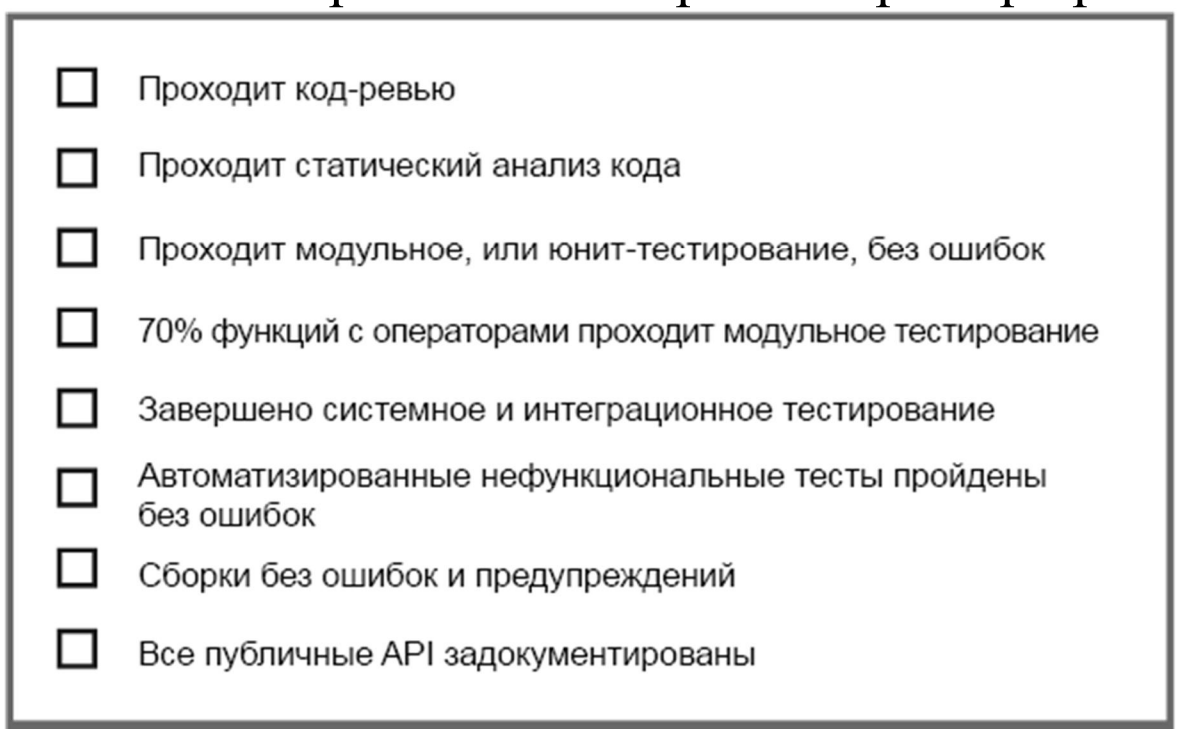


Рис. 11.2. Пример критериев готовности, которые определяют, когда задача в бэклоге считается выполненной

Командам нужно определить свои собственные критерии готовности с учетом факторов, соответствующих их обстоятельствам. Помимо факторов, показанных на рис. 11.2, критерии готовности могут включать в себя следующие:

- Владелец продукта принимает задание.
- Соответствует руководству по пользовательскому интерфейсу.
- Проходит приемочное тестирование.
- Проходит тестирование производительности.
- Проходит выбранные регрессионные тесты.
- Зарегистрировано изменение кода.
- Обновлена документация по требованиям.
- Проходит автоматическое сканирование на уязвимости.

Несколько версий критериев готовности

Командам понадобится более одной версии критериев готовности (Definition of Done) в двух общих случаях.

Несколько типов критериев готовности

Несколько типов критериев готовности бывают полезными или необходимыми при выполнении разных видов работ. Например, критерии готовности написания кода могут включать в себя полное регрессионное

тестирование, в то время как для написания пользовательской документации оно не нужно. Каждая версия критериев готовности должна представлять собой выходные критерии для перехода к работе над следующей задачей и гарантировать принцип, что никакой работы над задачей, удовлетворившей критериям готовности, больше не потребуется.

Несколько уровней критериев готовности

Второй случай, когда нужно несколько версий критериев готовности, — когда невозможно полностью выполнить работы в течение спринта. Например, при интеграции программного продукта в аппаратное решение критерии готовности первого уровня могут включать в себя прохождение тестов в условиях симуляции, но не обязательно их прохождение на целевом оборудовании, если оно еще не в наличии. Критерии готовности второго уровня будут включать в себя прохождение всех тестов уже на целевом оборудовании.

Также, если ваш программный продукт зависит от программного продукта другой команды или подрядчика и если вам еще не доставили необходимые компоненты, вы можете вывести критерии готовности первого уровня, обеспечив прохождение всех тестов посредством имитации недостающих объектов. Критерии готовности второго уровня будут удовлетворены при прохождении тестов с уже доставленными компонентами.

Несмотря на то что иногда существуют разумные причины на создание нескольких версий критериев готовности, оно может породить риск того, что «готово» не будет означать готовность в действительности, а это приведет к низкому качеству и накоплению дополнительной работы в промежутках между различными критериями. Лучше всего по возможности избегать этого.

Улучшение критериев готовности

В унаследованных системах распространенной проблемой является то, что унаследованные крупные кодовые базы невозможно мгновенно привести в вид, соответствующий критериям готовности. Таким образом, критерии готовности для унаследованной системы лучше занизить по сравнению с теми, которые применяются к новым системам. По мере улучшения качества унаследованного кода вы можете постепенно улучшать критерии готовности.

Распространенные проблемы при создании критериев готовности

Когда ваши команды определяют и выполняют критерии готовности, следует учитывать следующие распространенные проблемы:

- *Критерии готовности задают слишком низкий стандарт для качества релиза*. В подробностях критерии могут различаться, но суть критериев готовности должна состоять в том, что если работу признали готовой, то ее можно выпускать в релиз без дальнейших доработок.
- *Критериев готовности слишком много*. Список из 50 критериев готовности будет слишком громоздок, команде будет неудобно с ним работать, собственно, она и не будет.
- *Слишком высокие критерии готовности для унаследованных систем*. Старайтесь не создавать критериев готовности, которым невозможно следовать при работе с унаследованными системами, или высоких настолько, что объем работ будет слишком велик для соблюдения сроков, выделенных на проект.
- *Критерии готовности указывают на действия, а не факты*. Критерий вроде «провели разбор кода» указывает на действие. Критерий вроде «разбор кода проведен» указывает на факт.
- *Многоуровневые критерии готовности слишком нечеткие*. В первую очередь старайтесь вообще не применять многоуровневые критерии готовности. Если уж их применяете, убедитесь, что критерии для каждого уровня точно устанавливают соответствующую готовность.

Ключевой принцип: поддерживайте качество на уровне релиза

Критерии готовности применяются к отдельным элементам. Кроме того, непрерывное обеспечение качества уровня релиза для общей кодовой базы позволяет подстраховаться, что помогает получить эффективность во многих других практических методах, в том числе в написании кода, отладке и получении конструктивной обратной связи от пользователей.

Дисциплина, позволяющая часто приводить программный продукт к уровню качества релиза, приносит два важных преимущества.

Первое: поддержание качества на уровне релиза уменьшает промежуток между появлением и обнаружением недочета. Если вы приводите софт к уровню качества релиза каждые 1–3 недели, у вас никогда не возникнет длинного разрыва. Это позволяет обеспечить высокий уровень качества. Чем чаще ПО доводят до высокого уровня качества, тем легче его поддерживать на этом уровне и избежать накопления технического долга.

Второе: поддержка планирования и отслеживания. Если софт находится на уровне качества релиза к концу каждого спринта, значит, что над имеющейся функциональностью дальнейшей работы не требуется. Если ПО не находится на уровне качества релиза, значит, какое-то неопределенное количество дополнительной работы по улучшению качества придется выполнить позже. Работа по улучшению качества накапливается с каждым спринтом, а это ухудшает возможность определить действительное состояние проекта. Более подробно эта тенденция обсуждается в главе 11 «Еще более эффективное качество».

По обеим вышеуказанным причинам командам важно поддерживать качество продукта на уровне релиза к концу каждого без исключения спринта. Во многих случаях по завершении вы будете выпускать софт в продакшен. В некоторых случаях это неуместно — например, если вы работаете в регулируемой отрасли, релизы вашего ПО связаны с выпуском оборудования, или работа еще не переступила порог минимальной жизнеспособности продукта.

Уменьшайте количество доработок

«Доработка» означает работу над задачами, которые до этого уже объявили готовыми. Она включает в себя устранение ошибок, проработку непонятых требований, модификации тестовых сценариев и прочие исправления работы, которую нужно было выполнить правильно в первую очередь.

Доработка мешает ведению проектов, потому что влечет за собой непредсказуемый объем работ, в планах проекта на эту работу нет времени, притом она не приносит никакой дополнительной ценности.

Измерение объема доработок полезно в целях уменьшения их количества. Это обсуждается в главе 18 «Еще более эффективные измерения».

Прочие соображения

Парное программирование

Парное программирование — это практический метод, при котором два разработчика сидят плечом к плечу, один пишет код, а другой рецензирует код в реальном времени. Эти роли иногда рассматривают как «пилот и штурман». Парное программирование тесно связано с экстремальным программированием.

Данные о парном программировании, полученные в отрасли за долгие годы, показали, что отдача от двух специалистов, работающих в паре, примерно сопоставима с суммарной отдачей от двух специалистов, работающих по отдельности, — качество выше, а работа выполняется быстрее [Уильямс, 2002; Боэм, 2004].

Несмотря на то что парное программирование прочно ассоциируется с Agile, я бы не стал делать акцент на нем как на высокоэффективном практическом методе, потому что, по опыту, большая часть разработчиков предпочитает работать не в паре. В результате парное программирование в большинстве компаний прижилось как нишевый практический метод, выборочно применяемый главным образом в критических и сложных частях проектирования или написания кода. Если бы моя команда захотела применять парное программирование как можно чаще, я бы это поддержал, но не настаивал.

Моб-программирование и роение

Моб-программирование (моббинг) — это практический метод, при котором вся команда работает над одной и той же задачей одновременно на одном и том же компьютере.

Роение (swarming) — это когда вся команда работает над одной и той же историей одновременно, но каждый член команды занимается своей частью истории на своем компьютере. (Эти термины толкуются по-разному, поэтому, возможно, вы где-то слышали нечто отличное от сказанного мной здесь.)

Некоторым командам удавалось успешно применять эти методы, но вопрос об их эффективности остается открытым. Даже некоторые рецензенты этой книги советовали либо вообще их не применять, либо применять с новыми командами, либо применять только с опытными командами. Я не вижу какого-то центра притяжения для этих методов, поэтому в целом расцениваю моб-программирование и роение как нишевые практики, которые нужно задействовать выборочно или вовсе их не применять.

Рекомендации

Изучайте

- Проведите обзор ваших мероприятий по контролю качества, где и когда были обнаружены недочеты. Оцените способность Agile-методов обнаруживать больше недочетов как можно скорее.
- Проведите обзор неисправленных ошибок в ваших проектах. Сколько таких ошибок? Указывает ли это количество на то, что в ваших проектах бэклоги накапливают скрытые недочеты как они есть, без их устранения?
- Попросите команды показать вам свои критерии готовности. Есть ли у них ясно документированное определение? Применяется ли оно? Соответствуют ли критерии в совокупности качеству релиза?
- Узнайте, измеряют ли ваши команды процент задач, которые требуют доработки, в своих проектах и используют ли эти данные как основу для работы по улучшению процесса.
- Какие препятствия существуют между тем, чем команды занимаются сегодня, и тем, что нужно для качества релиза? Чем вы можете помочь своим командам, чтобы преодолеть эти препятствия?

Приспосабливайтесь

- На основе вашей оценки того, когда и где были выявлены дефекты, создайте план их предупреждения с помощью практических методов.
- Создайте план по сокращению неисправленных ошибок в ваших проектах и последующему сдерживанию их количества.
- Поработайте со своими командами, чтобы измерить процент задач, которые нужно доработать в проекте. Отслеживайте этот процент как часть работы, направленной на улучшение процесса.

- Устраните препятствия между тем, чем ваши команды занимаются сегодня, и тем, что нужно для качественного релиза.

Дополнительные ресурсы

[Стив Макконнелл, 2019]. Understanding Software Projects Lecture Series. Серия Construx OnDemand. Сетевой ресурс (2019 год). https://ondemand.construx.com. Эта серия содержит обширное обсуждение проблем, связанных с качеством.

[Майкл Т. Найгард, 2018]. Release It! Design and Deploy Production-Ready Software, 2nd Ed. Эта книга — актуальное и занимательное описание того, как проектировать и создавать системы высокого качества, в которой рассматриваются свойства, не относящиеся к функционалу, в том числе безопасность, стабильность, доступность, развертываемость и другие.

ГЛАВА 12. Еще более эффективное тестирование

Методология Agile изменила традиционный подход к тестам в четырех аспектах. Во-первых, разработчики должны больше времени уделять тестированию продукта. Во-вторых, акцент сместился на проведение раннего тестирования — функциональность тестируется сразу же после создания. В-третьих, больше внимания стало уделяться автоматизации тестирования. Наконец, подчеркивается, что тестирование является средством уточнения требований и совершенствования проектирования.

Эти четыре принципа позволяют подстраховаться при использовании некоторых практических Agile-методов, например при своевременном проектировании и реализации. Без подстраховки в виде комплексных автоматизированных тестовых наборов постоянно изменяющееся проектирование и код могут привести к огромной волне дефектов — многие из которых остаются незамеченными и попадают в пул скрытых дефектов, как рассказано в главе 11 «Еще более эффективное качество». Благодаря подстраховке в виде автотестов большинство дефектов обнаруживается сразу после их появления, что помогает достичь сокращения интервала между их появлением и обнаружением.

В последующих разделах мы расскажем о наиболее эффективных, по нашему мнению, практических методах тестирования для проектов, выполняемых по Agile.

Ключевой принцип: применяйте автотесты, созданные командой разработчиков

Команде разработчиков нужно написать автоматизированные тесты, которые встраиваются в автоматизированную систему сборки/развертывания. Лучше всего применять несколько слоев и видов

тестирования: тестирование API, юнит-тестирование, приемочное тестирование (UAT), тестирование пользовательского интерфейса, поддержку моков, случайные входные и выходные данные, симуляции и так далее.

Тесты пишет кросс-функциональная *команда*, в составе которой есть разработчики, тестировщики или бывшие тестировщики. Лучше всего, чтобы разработчики писали юнит-тесты до написания соответствующего кода. Разработка и автоматизация тестов — неотъемлемая часть реализации задач в бэклоге, которая также включена в оценку усилий.

Команде нужно поддерживать автоматизированную тестовую среду, которую можно запускать по необходимости. Сочетание автоматизированных юнит-тестов (тесты на уровне кода) и автоматизированного тестирования на уровне пользователя должно стать одним из основных критериев готовности.

Разработчики должны иметь способность проводить тесты на месте с помощью юнит-тестов и мокирования недостающих элементов. Разработчик должен иметь возможность запускать тестовые наборы для завершенного компонента продукта за несколько минут либо на общем сборочном сервере команды, либо на своем компьютере.

Локальный код подается в среду интеграции, где объединяются юнит-тесты разработчиков наряду со сборками. У команды должна быть возможность пройти тест полностью, в том числе все автоматизированные тесты на уровне юнитов и пользователя, за 1–2 часа. Во многих средах это время измеряется минутами. Тестирование нужно полностью проходить несколько раз в день.

При сложной организации разработчиков должна быть возможность поддерживать непрерывную интеграцию посредством запуска всех автоматизированных тестов каждый раз, когда регистрируется изменение кода. В крупных проектах это требует множества виртуальных сред, которые объединяют для параллельного запуска тестов, что, в свою очередь, требует отдельной команды (включающей тестировщиков), которая занимается созданием, поддержкой и расширением сервера непрерывной интеграции, собирая воедино тесты, полученные от разных команд.

Крупные компании, вроде *Amazon*и *Netflix*, могут поддерживать быстрое непрерывное тестирование, потому что у них есть команды, которые занимаются исключительно этим. Такие компании вложили немалые средства в оборудование и долгие годы совершенствовались. Компании, которые только начали внедрять непрерывную интеграцию, далеки от потребностей *Amazon* или *Netflix* и должны рассчитывать на результат соответственно своим запросам.

Автоматизация тестов в унаследованных системах

Неспособность разработать совершенные тесты не должна быть поводом для отказа от создания автотестов. Нам встречались команды, которые наследовали кодовые базы низкого качества, внедряли базовые поверхностные тесты, медленно пропускали код через автотесты и получали значительную пользу даже при таком слабом уровне автоматизации. Вы можете так сделать с помощью понижения планки критериев готовности, которую со временем нужно будет поднимать.

Лучше всего это помогает сосредоточить тестирование унаследованной системы на участках кода, над которыми активнее всего ведется работа. Мало пользы в тестировании кода, который стабилен уже не первый год.

Больше ключевых принципов для эффективного тестирования в Agile

Помимо включения тестировщиков в команды разработчиков и применения автоматизированного тестирования помните о следующих ключевых принципах для эффективного тестирования в Agile.

Разработчики несут основную ответственность за тестирование своего кода

Интеграция тестировщиков в команды разработчиков может непреднамеренно привести к тому, что разработчики не тестируют свой код — наперекор задуманному! Разработчики несут основную ответственность за качество своей работы, в том числе тестирования. Остерегайтесь следующих тревожных звоночков:

- Задачи из бэклога выполняются только ближе к концу каждого спринта (это означает, что тестирование проводят после написания кода или отдельно).
- Разработчики переходят к написанию кода для новой задачи до того, как приведут предыдущие задачи к критериям готовности.

Измеряйте покрытие кода

Написание тестовых сценариев до написания кода («сначала тесты») может быть полезной дисциплиной, но мы обнаружили, что для новых кодовых баз важнее измерение покрытия кода при юнит-тестировании в сочетании с последующей автоматизацией тестирования. При юнит-тестировании покрытие кода в 70% — это хороший практический уровень, к которому нужно стремиться при написании нового кода. Покрытие кода в 100% при юнит-тестировании встречается редко и, как правило, затрачивает больше усилий, чем приносит пользы (разумеется, за исключением продуктов, где безопасность системы критически важна).

Организации, с которыми работала моя компания, в лучшем случае приближались к соотношению тестового кода и окончательного кода примерно на уровне 1 : 1, что включает в себя код юнит-тестов и код тестов более высокого уровня. Опять же это зависит от типа ПО. Стандарты ПО, где безопасность критически важна, отличаются от стандартов для делового или развлекательного ПО.

Остерегайтесь злоупотребления измерением покрытия кода тестами

Мы обнаружили, что показателем покрытия в «заявленные 70%» склонны злоупотреблять гораздо чаще, чем того можно ожидать.

Нам встречались команды, которые дезактивировали проваленные тестовые сценарии, чтобы завысить процент прохождения, либо создавали тестовые сценарии, прохождение которых всегда успешно.

В таких случаях полезнее исправить систему, а не человека. Такое поведение означает, что команды ставят разработку в больший приоритет, чем тестирование. Менеджерам требуется донести, что тестирование и контроль качества не менее важны, чем написание кода. Помогите своим командам осознать цель и пользу тестирования и обратите внимание на то, что 70% — это просто показатель, но не самоцель.

Отслеживайте статические показатели кода

Покрытие кода и другие тестовые показатели полезны, но не могут рассказать о качестве все. Статические показатели качества кода также важны: уязвимости безопасности, цикломатическая сложность, глубина вложенности решений, число стандартных параметров, размер файла, размер папки, длина рутинной операции, использование магических чисел, встроенный SQL, дублированный или копированный код, качество комментариев, соблюдение стандартов написания кода и так далее. Показатели дают подсказки о том, какие области кода может понадобиться проработать для поддержания качества.

Старательно пишите тестовый код

Тестовый код должен соответствовать тем же стандартам качества, что и окончательный код. При его написании нужно использовать понятные именования, избегать магических чисел и дубликатов, осуществлять должную проработку, применять единообразное форматирование, вести контроль версий и так далее.

Ставьте в приоритет поддержание качества тестовых наборов

Тесты имеют свойство ухудшаться со временем, и нередко встречаются наборы, в которых отключена большая доля тестов. Команде требуется

расценить обзор и поддержание качества тестовых наборов как неотъемлемую издержку проводимой разработки, а работу над тестами включить в критерии готовности. Это необходимо для постоянного поддержания качества ПО на уровне релиза — важно не позволять дефектам выходить из-под контроля.

Создайте отдельную структуру для написания и поддержки тестов

Если в вашей компании уже есть отдельная структура, ответственная за тестирование, полезно сделать ее основной ответственностью создание и поддержание качества приемочных тестов. Команда разработчиков будет по-прежнему писать и запускать приемочные тесты — это важно для уменьшения интервала между появлением и обнаружением недочета. Но она будет нести ответственность за такую работу уже во вторую очередь.

Часто встречается, что приемочные тесты выполняются в отдельной QA-среде. Это полезно, когда содержание среды интеграции постоянно изменчиво, — отдельная среда может быть стабильнее.

Не зацикливайтесь на юнит-тестах

В Agile при тестировании есть риск преувеличивать значение юнит-тестов и недооценивать важность тестов таких независимых свойств, как масштабируемость, производительность и им подобных, которые становятся очевидными при запуске интеграционных тестов для более крупных программных систем. Убедитесь в том, что было проведено тестирование всей системы целиком, прежде чем команда заявит о выполнении задач спринта.

Прочие соображения

Ручное/исследовательское тестирование

Тестирование вручную остается актуальным в форме исследовательского тестирования, тестирования на удобство использования и прочих видов тестов.

Современный подход к тестированию

Мир разработки находится в эпицентре кардинальных изменений в подходах к тестированию, которые стали возможны благодаря облачным вычислениям, легкости внесения и отката изменений и в некоторых случаях новым видам багов, связанных с облачными вычислениями. Если ваше понимание практических методов тестирования основано на привычном софте и не обновлялось, потратьте некоторое время на изучение современных методов тестирования, например релизов Canary

(A/B-тестирование), Chaos Monkey/Simian Army и прочих методов, основанных на облачных решениях.

Рекомендации

Изучайте

- Проведите обзор, используют ли ваши команды подходы к автоматизированному тестированию, которые включают стандарты по покрытию тестируемого кода и минимально допустимое покрытие кода при тестировании.
- Определите тесты, которые вашим командам нужно выполнять вручную. Нужен ли вашей команде план по автоматизированию каких-либо из ручных тестов?

Приспосабливайтесь

- Определите уровень цели для автоматизации тестирования в каждом из типов ваших проектов. Создайте план достижения этого уровня на следующие пару десятков лет.

Дополнительные ресурсы

[Лиза Криспин и Джанет Грегори, 2009]. Agile Testing: A Practical Guide for Testers and Agile Teams. Это популярный справочник о том, чем тестирование отличается в командах и проектах, где применяется Agile.

[Николь Форсгрен и др., 2018]. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. Эта книга излагает текущие данные по самым эффективным практическим методам тестирования Agile.

[Дженни Стюарт и Мелвин Перес, 2018]. Retrofitting Legacy Systems with Unit Tests. Сетевой ресурс (июль 2018 года). В этой статье рассматриваются особые проблемы, возникающие при тестировании унаследованных систем.

[Майкл К. Физерс, 2004]. Эффективная работа с унаследованным кодом (Вильямс, 2016). Эта книга подробно рассказывает о работе с унаследованными системами, в том числе о тестировании.

ГЛАВА 12. Еще более эффективное тестирование

Методология Agile изменила традиционный подход к тестам в четырех аспектах. Во-первых, разработчики должны больше времени уделять тестированию продукта. Во-вторых, акцент сместился на проведение раннего тестирования — функциональность тестируется сразу же после создания. В-третьих, больше внимания стало уделяться автоматизации тестирования. Наконец, подчеркивается, что тестирование является средством уточнения требований и совершенствования проектирования.

Эти четыре принципа позволяют подстраховаться при использовании некоторых практических Agile-методов, например при своевременном

проектировании и реализации. Без подстраховки в виде комплексных автоматизированных тестовых наборов постоянно изменяющееся проектирование и код могут привести к огромной волне дефектов — многие из которых остаются незамеченными и попадают в пул скрытых дефектов, как рассказано в главе 11 «Еще более эффективное качество». Благодаря подстраховке в виде автотестов большинство дефектов обнаруживается сразу после их появления, что помогает достичь сокращения интервала между их появлением и обнаружением.

В последующих разделах мы расскажем о наиболее эффективных, по нашему мнению, практических методах тестирования для проектов, выполняемых по Agile.

Ключевой принцип: применяйте автотесты, созданные командой разработчиков

Команде разработчиков нужно написать автоматизированные тесты, которые встраиваются в автоматизированную систему сборки/развертывания. Лучше всего применять несколько слоев и видов тестирования: тестирование API, юнит-тестирование, приемочное тестирование (UAT), тестирование пользовательского интерфейса, поддержку моков, случайные входные и выходные данные, симуляции и так далее.

Тесты пишет кросс-функциональная *команда*, в составе которой есть разработчики, тестировщики или бывшие тестировщики. Лучше всего, чтобы разработчики писали юнит-тесты до написания соответствующего кода. Разработка и автоматизация тестов — неотъемлемая часть реализации задач в бэклоге, которая также включена в оценку усилий.

Команде нужно поддерживать автоматизированную тестовую среду, которую можно запускать по необходимости. Сочетание автоматизированных юнит-тестов (тесты на уровне кода) и автоматизированного тестирования на уровне пользователя должно стать одним из основных критериев готовности.

Разработчики должны иметь способность проводить тесты на месте с помощью юнит-тестов и мокирования недостающих элементов. Разработчик должен иметь возможность запускать тестовые наборы для завершенного компонента продукта за несколько минут либо на общем сборочном сервере команды, либо на своем компьютере.

Локальный код подается в среду интеграции, где объединяются юнит-тесты разработчиков наряду со сборками. У команды должна быть возможность пройти тест полностью, в том числе все автоматизированные тесты на уровне юнитов и пользователя, за 1–2 часа. Во многих средах это время измеряется минутами. Тестирование нужно полностью проходить несколько раз в день.

При сложной организации разработчиков должна быть возможность поддерживать непрерывную интеграцию посредством запуска всех автоматизированных тестов каждый раз, когда регистрируется изменение кода. В крупных проектах это требует множества виртуальных сред, которые объединяют для параллельного запуска тестов, что, в свою очередь, требует отдельной команды (включающей тестировщиков), которая занимается созданием, поддержкой и расширением сервера непрерывной интеграции, собирая воедино тесты, полученные от разных команд.

Крупные компании, вроде *Amazon*и *Netflix*, могут поддерживать быстрое непрерывное тестирование, потому что у них есть команды, которые занимаются исключительно этим. Такие компании вложили немалые средства в оборудование и долгие годы совершенствовались. Компании, которые только начали внедрять непрерывную интеграцию, далеки от потребностей *Amazon* или *Netflix* и должны рассчитывать на результат соответственно своим запросам.

Автоматизация тестов в унаследованных системах

Неспособность разработать совершенные тесты не должна быть поводом для отказа от создания автотестов. Нам встречались команды, которые наследовали кодовые базы низкого качества, внедряли базовые поверхностные тесты, медленно пропускали код через автотесты и получали значительную пользу даже при таком слабом уровне автоматизации. Вы можете так сделать с помощью понижения планки критериев готовности, которую со временем нужно будет поднимать.

Лучше всего это помогает сосредоточить тестирование унаследованной системы на участках кода, над которыми активнее всего ведется работа. Мало пользы в тестировании кода, который стабилен уже не первый год.

Больше ключевых принципов для эффективного тестирования в Agile

Помимо включения тестировщиков в команды разработчиков и применения автоматизированного тестирования помните о следующих ключевых принципах для эффективного тестирования в Agile.

Разработчики несут основную ответственность за тестирование своего кода

Интеграция тестировщиков в команды разработчиков может непреднамеренно привести к тому, что разработчики не тестируют свой код — наперекор задуманному! Разработчики несут основную

ответственность за качество своей работы, в том числе тестирования. Остерегайтесь следующих тревожных звоночков:

- Задачи из бэклога выполняются только ближе к концу каждого спринта (это означает, что тестирование проводят после написания кода или отдельно).
- Разработчики переходят к написанию кода для новой задачи до того, как приведут предыдущие задачи к критериям готовности.

Измеряйте покрытие кода

Написание тестовых сценариев до написания кода («сначала тесты») может быть полезной дисциплиной, но мы обнаружили, что для новых кодовых баз важнее измерение покрытия кода при юнит-тестировании в сочетании с последующей автоматизацией тестирования. При юнит-тестировании покрытие кода в 70% — это хороший практический уровень, к которому нужно стремиться при написании нового кода. Покрытие кода в 100% при юнит-тестировании встречается редко и, как правило, затрачивает больше усилий, чем приносит пользы (разумеется, за исключением продуктов, где безопасность системы критически важна).

Организации, с которыми работала моя компания, в лучшем случае приближались к соотношению тестового кода и окончательного кода примерно на уровне 1 : 1, что включает в себя код юнит-тестов и код тестов более высокого уровня. Опять же это зависит от типа ПО. Стандарты ПО, где безопасность критически важна, отличаются от стандартов для делового или развлекательного ПО.

Остерегайтесь злоупотребления измерением покрытия кода тестами

Мы обнаружили, что показателем покрытия в «заявленные 70%» склонны злоупотреблять гораздо чаще, чем того можно ожидать.

Нам встречались команды, которые дезактивировали проваленные тестовые сценарии, чтобы завысить процент прохождения, либо создавали тестовые сценарии, прохождение которых всегда успешно.

В таких случаях полезнее исправить систему, а не человека. Такое поведение означает, что команды ставят разработку в больший приоритет, чем тестирование. Менеджерам требуется донести, что тестирование и контроль качества не менее важны, чем написание кода. Помогите своим командам осознать цель и пользу тестирования и обратите внимание на то, что 70% — это просто показатель, но не самоцель.

Отслеживайте статические показатели кода

Покрытие кода и другие тестовые показатели полезны, но не могут рассказать о качестве все. Статические показатели качества кода также

важны: уязвимости безопасности, цикломатическая сложность, глубина вложенности решений, число стандартных параметров, размер файла, размер папки, длина рутинной операции, использование магических чисел, встроенный SQL, дублированный или копированный код, качество комментариев, соблюдение стандартов написания кода и так далее. Показатели дают подсказки о том, какие области кода может понадобиться проработать для поддержания качества.

Старательно пишите тестовый код

Тестовый код должен соответствовать тем же стандартам качества, что и окончательный код. При его написании нужно использовать понятные именования, избегать магических чисел и дубликатов, осуществлять должную проработку, применять единообразное форматирование, вести контроль версий и так далее.

Ставьте в приоритет поддержание качества тестовых наборов

Тесты имеют свойство ухудшаться со временем, и нередко встречаются наборы, в которых отключена большая доля тестов. Команде требуется расценить обзор и поддержание качества тестовых наборов как неотъемлемую издержку проводимой разработки, а работу над тестами включить в критерии готовности. Это необходимо для постоянного поддержания качества ПО на уровне релиза — важно не позволять дефектам выходить из-под контроля.

Создайте отдельную структуру для написания и поддержки тестов

Если в вашей компании уже есть отдельная структура, ответственная за тестирование, полезно сделать ее основной ответственностью создание и поддержание качества приемочных тестов. Команда разработчиков будет по-прежнему писать и запускать приемочные тесты — это важно для уменьшения интервала между появлением и обнаружением недочета. Но она будет нести ответственность за такую работу уже во вторую очередь.

Часто встречается, что приемочные тесты выполняются в отдельной QA-среде. Это полезно, когда содержание среды интеграции постоянно изменчиво, — отдельная среда может быть стабильнее.

Не зацикливайтесь на юнит-тестах

В Agile при тестировании есть риск преувеличивать значение юнит-тестов и недооценивать важность тестов таких независимых свойств, как масштабируемость, производительность и им подобных, которые становятся очевидными при запуске интеграционных тестов для более крупных программных систем. Убедитесь в том, что было проведено

тестирование всей системы целиком, прежде чем команда заявит о выполнении задач спринта.

Прочие соображения

Ручное/исследовательское тестирование

Тестирование вручную остается актуальным в форме исследовательского тестирования, тестирования на удобство использования и прочих видов тестов.

Современный подход к тестированию

Мир разработки находится в эпицентре кардинальных изменений в подходах к тестированию, которые стали возможны благодаря облачным вычислениям, легкости внесения и отката изменений и в некоторых случаях новым видам багов, связанных с облачными вычислениями. Если ваше понимание практических методов тестирования основано на привычном софте и не обновлялось, потратьте некоторое время на изучение современных методов тестирования, например релизов Canary (A/B-тестирование), Chaos Monkey/Simian Army и прочих методов, основанных на облачных решениях.

Рекомендации

Изучайте

- Проведите обзор, используют ли ваши команды подходы к автоматизированному тестированию, которые включают стандарты по покрытию тестируемого кода и минимально допустимое покрытие кода при тестировании.
- Определите тесты, которые вашим командам нужно выполнять вручную. Нужен ли вашей команде план по автоматизированию каких-либо из ручных тестов?

Приспосабливайтесь

- Определите уровень цели для автоматизации тестирования в каждом из типов ваших проектов. Создайте план достижения этого уровня на следующие пару десятков лет.

Дополнительные ресурсы

[Лиза Криспин и Джанет Грегори, 2009]. Agile Testing: A Practical Guide for Testers and Agile Teams. Это популярный справочник о том, чем тестирование отличается в командах и проектах, где применяется Agile.

[Николь Форсгрен и др., 2018]. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. Эта книга излагает текущие данные по самым эффективным практическим методам тестирования Agile.

[Дженни Стюарт и Мелвин Перес, 2018]. Retrofitting Legacy Systems with Unit Tests. Сетевой ресурс (июль 2018 года). В этой статье

рассматриваются особые проблемы, возникающие при тестировании унаследованных систем.

[Майкл К. Физерс, 2004]. Эффективная работа с унаследованным кодом (Вильямс, 2016). Эта книга подробно рассказывает о работе с унаследованными системами, в том числе о тестировании.

ГЛАВА 13. Еще более эффективное создание требований

Каждое исследование, которое я видел первые 25 лет моей работы в разработке, рассматривало причины трудностей, возникающих во время проекта, и главной причиной проблем авторы называли низкокачественные требования — неполные, некорректные, противоречивые и так далее. За последние 10 лет наиболее частой трудностью, которая мне встречалась при работе над Agile-проектами, было исполнение роли владельца продукта. И причина этому, как вы уже догадались, — требования.

Поскольку требования считаются распространенным и постоянным источником трудностей в разработке ПО, в следующих двух главах я собираюсь глубже окунуться в их специфику.

Жизненный цикл требований в Agile

В сравнении с временами 25-летней давности сегодня у нас есть эффективные методы создания требований, которые можно использовать при разработке проектов по Agile. Эти практические методы помогают осуществлять следующие мероприятия по разработке согласно основным требованиям:

- *Выявление* — первоначальное обнаружение требований.
- *Анализ* — развитие более полного и ясного понимания требований, в том числе их приоритетов.
- *Спецификация* — выражение требований в неизменном виде.
- *Валидация* — гарантия того, что требования корректны (будут удовлетворять потребностям клиента) и они правильно сформулированы.

Для большинства технических методов нет большой разницы, где они будут применяться, — в Agile или в последовательной разработке. Отличается именно то, *когда* эти методы применяются командой.

В этой главе рассказывается о мероприятиях по выявлению и спецификации, а также затрагиваются азы анализа. В следующей главе основное внимание уделяется приоритету анализа. Главные технические методы валидации требований в Agile-проектах включают в себя непрерывные обсуждения требований и обзор итогов спринта в конце спринтов (то есть демонстрацию работающего ПО).

Чем отличаются требования Agile?

В проектах, выполняемых по Agile, работа над требованиями проводится не тогда же, когда она делается при последовательной разработке. На рис. 13.1 показано это различие.

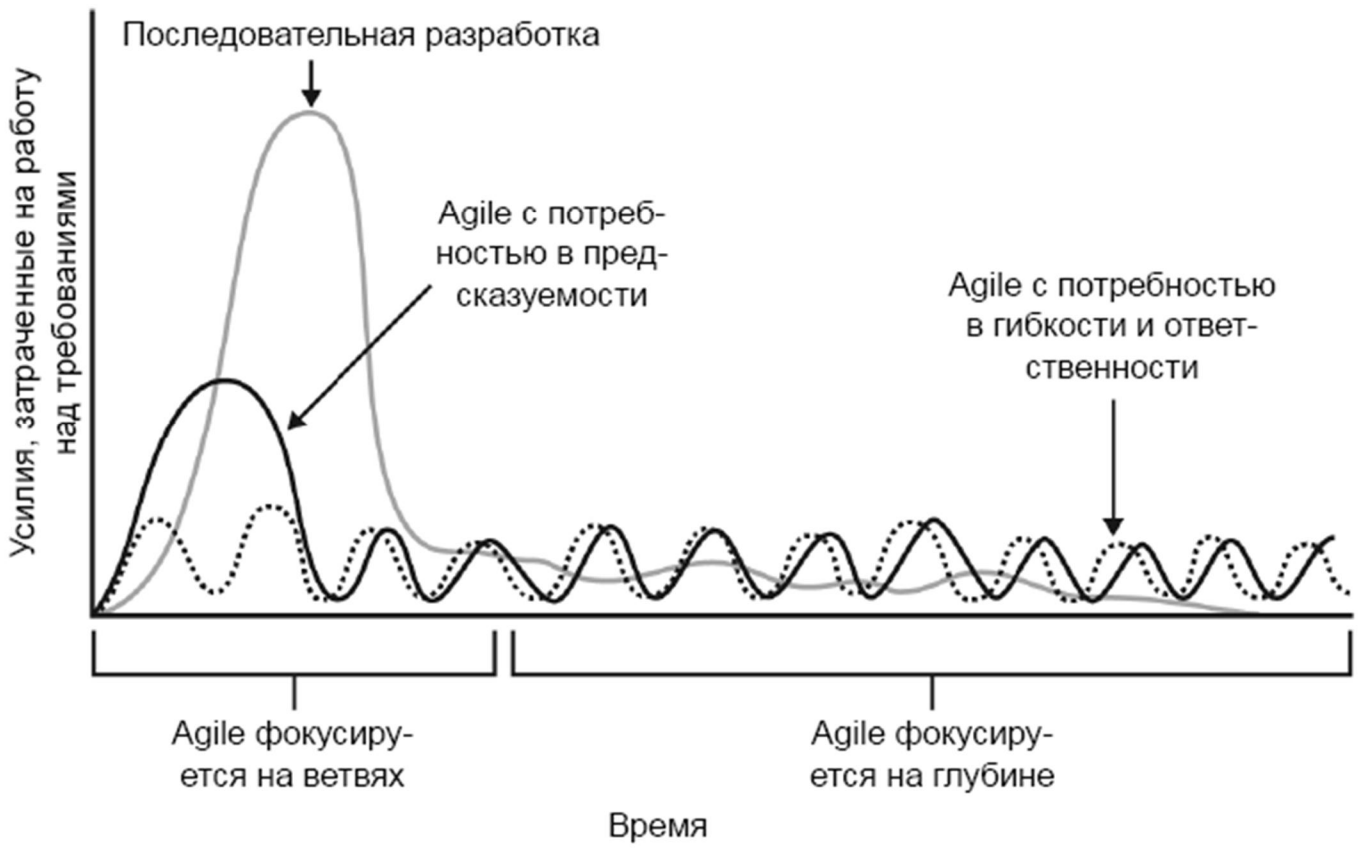


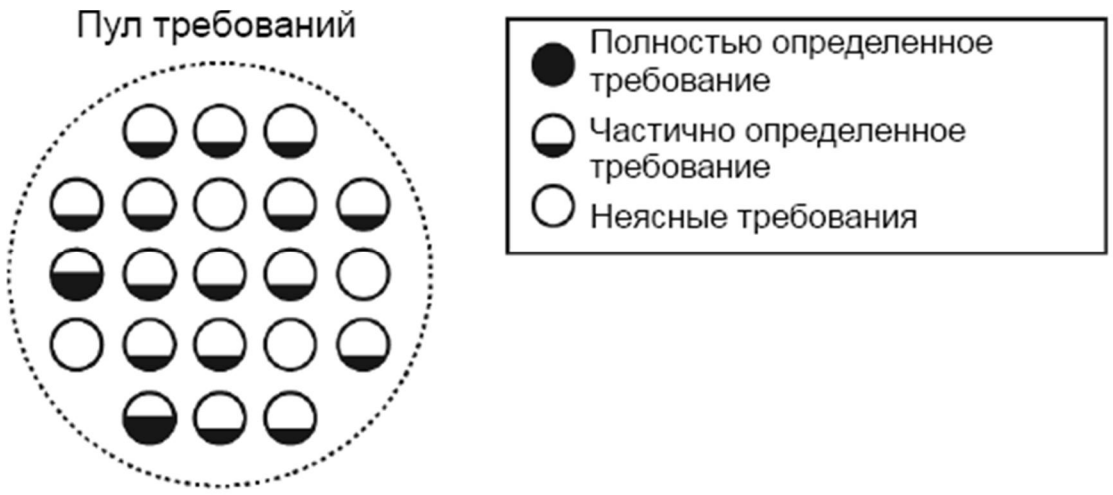
Рис. 13.1. Различия в заблаговременной работе над требованиями справедливы как для Agile-проектов, в которых требуется предсказуемость или гибкость, так и для проектов, выполняемых по модели последовательной разработки. Адаптация исследований [Вигерс, 2013]

В проектах последовательной разработки значительная часть работы над требованиями выполняется крупными партиями в начале проекта. Предварительные работы в более мелких Agile-проектах сосредоточены в основном на осознании объема требований.

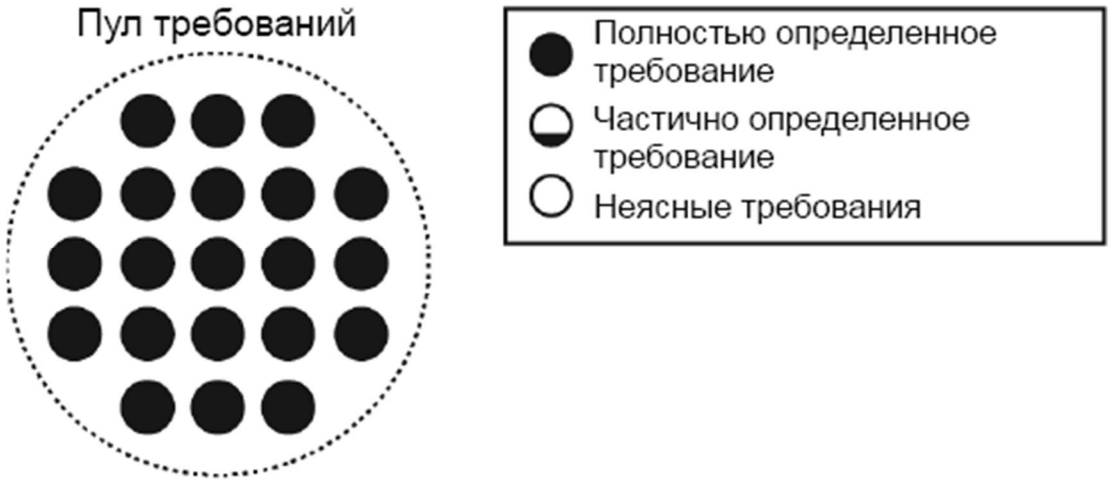
В Agile-проектах, требующих предсказуемости, работают над требованиями раньше, чем при выполнении прочих Agile-проектов. Уточнение подробностей каждого требования (проработка) в обоих случаях откладывается на срок незадолго до того, как начнется разработка по этим требованиям.

При выполнении Agile-проектов команды нацелены на определение сути каждого требования в начале проекта, оставляя большую часть детальной проработки, чтобы заняться ею в срок, а в некоторых случаях и вовсе оставляют всю работу на выполнение в выделенный срок. В Agile-проектах требования не прорабатываются меньше — они прорабатываются позже. При выполнении некоторых проектов допускают ошибку — не проводят проработку, но это больше характерно для процессов, где нужно сначала писать код, а потом устранять ошибки, но не для Agile. В еще более эффективных Agile-проектах прорабатывают требования с помощью практических методов, о которых будет рассказано далее.

Приведем наглядное представление о том, как при выполнении проектов по Agile воспринимаются требования:



В проектах последовательной разработки подробности каждого требования прорабатываются заранее, лишь небольшое количество работы по требованиям остается на более поздние этапы проекта. Требование разрабатывается заранее полностью, а не только его суть, как это показано ниже:



Работа над подробностями требований выполняется и в Agile, и в последовательной разработке, но она происходит в разное время.

Это ведет к тому, что при выполнении проекта разные виды работ выполняются в разное время, как это показано на рис. 13.2.

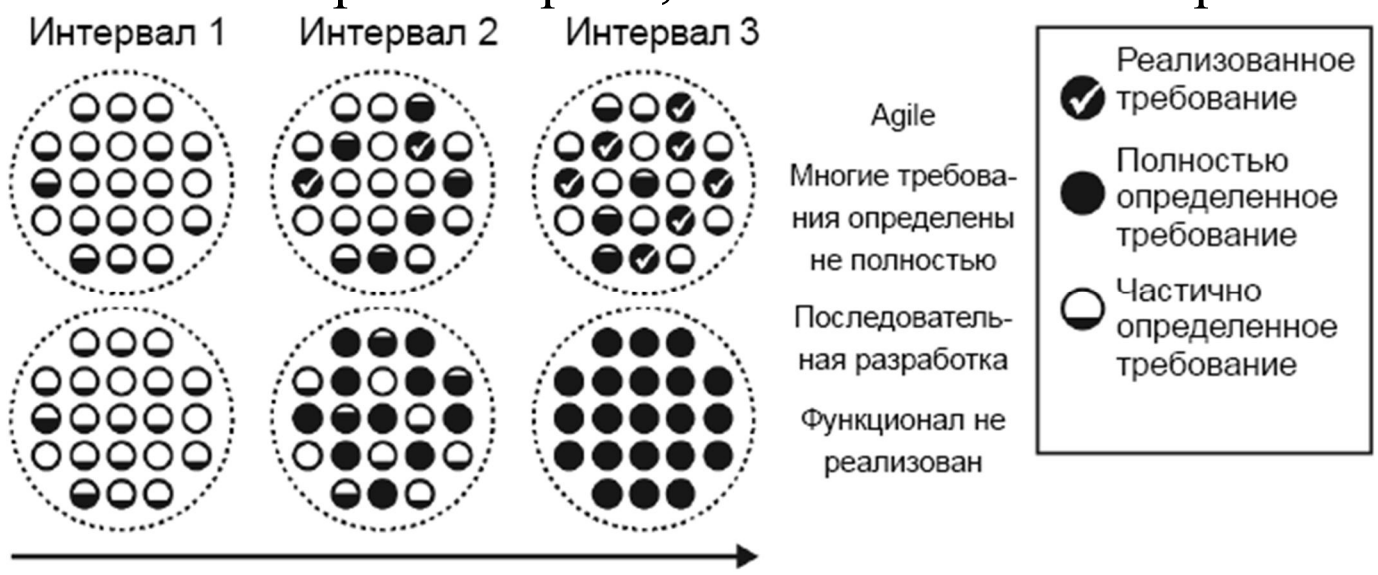


Рис. 13.2. Разница в степени завершенности требований и завершенности функционала проектов в разные отрезки времени при разработке по Agile и при последовательной разработке

Выполняя большую часть работы над требованиями заблаговременно, сторонники модели последовательной разработки объясняют суть так: «Ставлю на то, что заблаговременная проработка требований добавит ценность на последующих этапах проекта, позволит снизить неопределенность и что коэффициент порчи такой заблаговременной работы будет в пределах разумного» («порча» означает, что работа над

требованиями становится неактуальной ко времени начала работ над реализацией).

Сторонники Agile объясняют суть иначе: «Ставлю на то, что выполнение работ по сквозной реализации (не только работа над требованиями) и обратная связь позволят снизить неопределенность. Ставлю на то, что выполнение большого количества работы над требованиями заблаговременно предусмотрит много подробностей, которые потом станут ненужными к моменту начала реализации. Издержки от порчи требований будут выше, чем любая добавленная ценность, которая может быть получена при полном заблаговременном определении требований».

Доля правды присутствует в обоих утверждениях, поскольку то, какой подход работает лучше, определяется контекстом по Cynefin — «сложные» или «запутанные», уровнем навыков людей, занимающихся требованиями, и тем, насколько команда уверена в том, что работает в правильном контексте — «сложные» или «запутанные».

Cynefin и работа над требованиями

Для «сложных» проблем по Cynefin можно заранее смоделировать полностью всю систему, если команда достаточно квалифицированна для разработки требований.

Для «запутанных» проблем по Cynefin невозможно заранее узнать, что собой будет представлять система в итоге. Разработка требований — это процесс, требующий обучения как от команды разработчиков, так и от бизнеса. В этом контексте даже выококвалифицированные люди не знают всех подробностей того, что им нужно, до тех пор пока не попробуют это сделать.

Попытки определить требования заблаговременно при решении «запутанных» проблем сопровождаются следующими трудностями:

- *Изменение требований*, их нужно проработать заново в промежуток времени между изначальной проработкой требований и началом работ по реализации. Исходная проработка в итоге окажется пустой тратой времени.
- *Требования снимают* после того, как они в значительной мере были уже проработаны. Проработка прошла напрасно.
- *Выполняются требования, которые не нужны на самом деле*, что выясняется после того, как рабочее программное обеспечение попадает к конечному пользователю.
- *Некоторые требования по ходу проекта пропускаются или появляются*, что ведет к проблемам в проектировании и реализации при подходах, которые предполагают, что все требования были определены полностью или почти полностью заблаговременно. Работы по проектированию и реализации прошли даром.

На рис. 13.3 показаны издержки, возникающие при изменении требований после того этапа проекта, на котором требования при последовательной разработке должны быть полностью определены с помощью интервала 3 из рис. 13.2.

В верхней части рисунка, который относится к Agile-проекту, конечно, нет издержек в неизменных требованиях. Издержки из-за измененных или удаленных требований различаются в зависимости от степени выполнения работы. Частично определенные требования в результате дают уменьшение издержек (обратите внимание на точки в частично заполненных кружочках), а полностью определенные требования приводят к большим издержкам.

В нижней части рисунка, который относится к проекту по модели последовательной разработки, видна более высокая доля издержек при изменении и удалении требований по причине более высокого вложения средств и сил в проработку этих требований до их удаления или изменения.

Фальстарты и тупики все-таки случаются и при проработке требований в Agile, но меньшее вложение сил и средств в такие фальстарты и тупики ведет к меньшим потерям в общем.

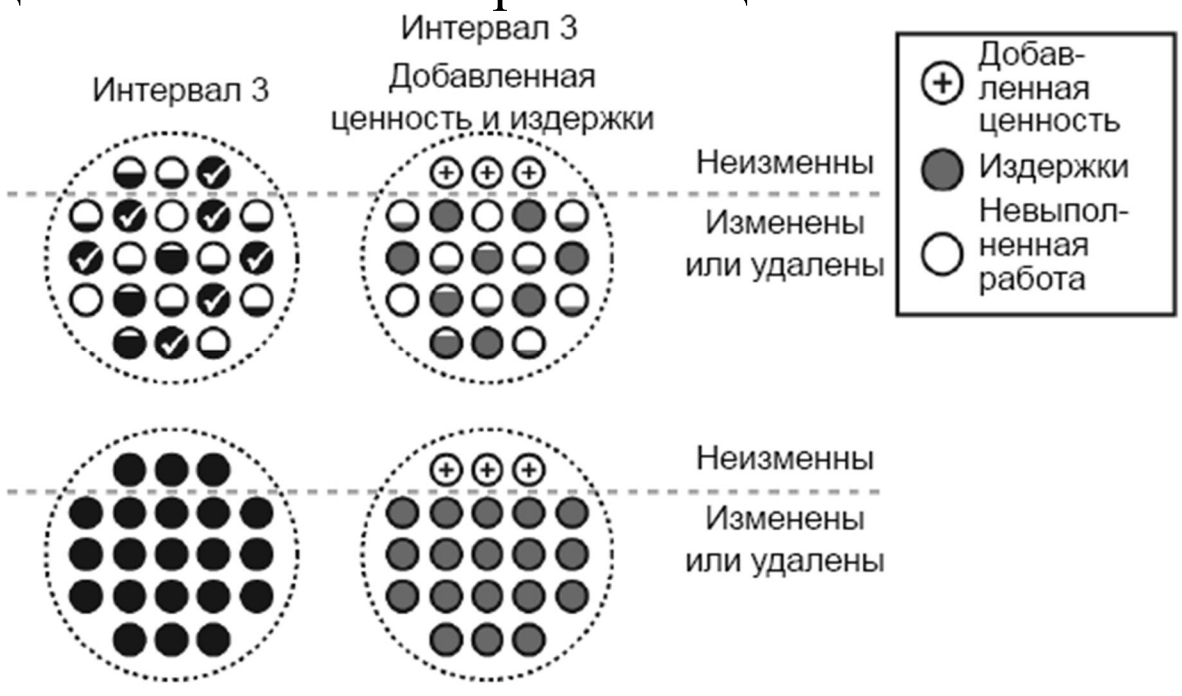


Рис. 13.3. Различия в издержках при изменении требований в середине Agile-проекта и проекта по модели последовательной разработки

Требования Agile: истории

Требования Agile наиболее часто выражены в форме историй, которые выглядят примерно так:

Как <тип пользователя>, я хочу <цель/желание>, чтобы<польза>

История — это ограниченный, определенный набор функциональности. Не все истории представляют собой требования. Некоторые примеры приведены в табл. 13.1.

Таблица 13.1. Примеры пользовательских историй

	Тип пользователя		Цель/Желание		Польза
Как	руководитель разработки	**я хочу**	количественно узнать статус своего проекта,	**чтобы**	я мог постоянно информировать остальную часть компании
Как	бизнес-лидер	**я хочу**	видеть статус всех проектов в одном месте так,	**чтобы**	мне было понятно, какому проекту стоит уделить внимание
Как	технический специалист	**я хочу**	сделать доклад о статусе как можно проще,	**чтобы**	я мог бˆольшую часть времени потратить на технические работы

При выполнении проектов по Agile специалисты часто полагаются на истории как на первичное средство, выражающее требования. Истории можно изобразить любыми способами, принятыми в Agile, — в электронной таблице, на карточках, стикерах. Как видно из примеров в таблице, истории не достаточно подробны для того, чтобы поддерживать процесс разработки только с помощью них. Истории — это лишь документированный и отслеживаемый шаблон для обсуждения между бизнесом и техническими специалистами.

Истории уточняются в разговорах, где обсуждают перспективы бизнеса, разработки и тестирования и другие перспективы — если это уместно.

Контейнер требований в Agile: бэклог продукта

В Agile требования, как правило, прописаны в бэклоге продукта, который содержит истории, эпики, темы, инициативы, функции, свойства, требования, улучшения и исправления — все необходимое для определения оставшегося объема проекта. Бэклог — стандартное понятие из Скрама. Команды, работающие по *Kanban*, могут называть его «входной очередью», но концепции схожи. Командам, работающим в условиях жесткого регулирования, могут быть нужны более формальные контейнеры (документы).

Большинство команд считает, что запас уточненных задач бэклога примерно на два спринта помимо текущего содержит достаточно подробностей для обеспечения планирования рабочего процесса и технической реализации.

Для команд, которые работают в основном в контексте Cynefin «Запутанные», может оказаться более практичным более короткий интервал планирования.

На рис. 13.4 показано, каким образом уточняются задачи бэклога продукта по мере приближения реализации этих задач. Я изобразил бэклог как воронку, внизу которой сосредоточена краткосрочная работа (Agile-команды, как правило, представляют бэклог как очередь, где краткосрочная работа сосредоточена вверху).

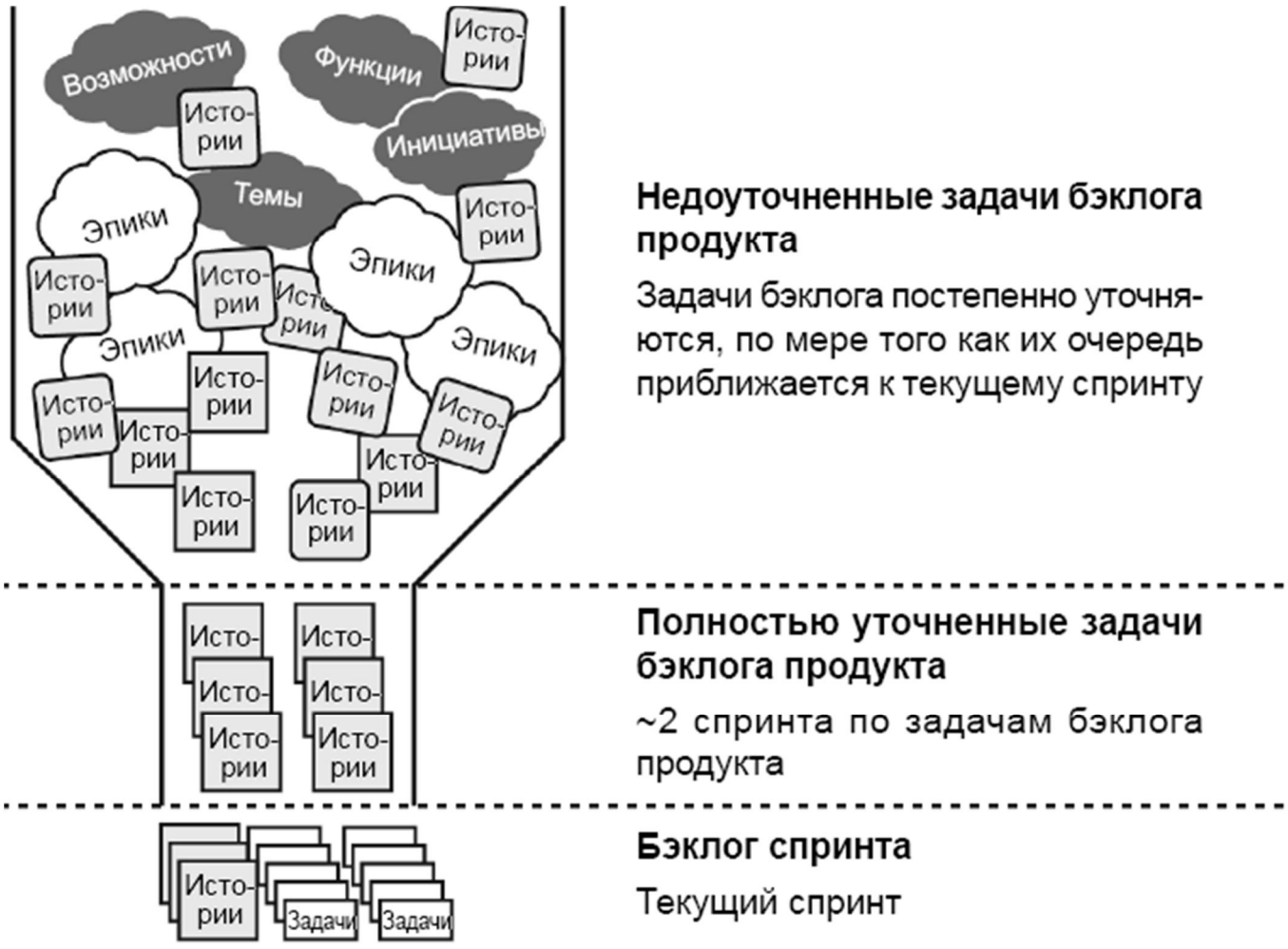


Рис. 13.4. Этапы при своевременном уточнении продукта бэклога в Agile

Что попадает в бэклог продукта?

В общем, бэклог продукта содержит требования, но четкого определения понятия «требование» нет. Наиболее распространенные задачи бэклога продукта выглядят так:

- *Требования* — общее понятие, которое включает функции, эпики, истории, исправления, улучшения и так далее. «Требование» не обязательно означает «полностью сформулированное, строгое требование». Как уже говорилось ранее в этой главе и как показано на рис. 13.2, большинство требований при работе по Agile находятся в частично определенном состоянии, до тех пор пока их не реализуют.
- *Функция* — инкремент функциональности, который доставляет ценность или расширяет возможности бизнеса. Под «функцией» подразумевается работа, на которую нужно затратить более одного спринта. Ее часто описывают как коллекцию пользовательских историй.
- *Эпик* — это история, на реализацию которой нужно затратить более одного спринта. Не существует всеобщего единого мнения по поводу того, что в точности можно считать эпиком или функцией, но в одном специалисты уверены точно — и то и другое слишком крупное, чтобы уложиться в один спринт.
- *Тема, тема для инвестиции, способности, инициатива, улучшение*и сходные понятия — поскольку ожидается, что бэклог будут уточнять, в него могут добавлять относительно аморфные задачи, особенно в самом конце бэклога, с пониманием того, что эти задачи еще будут уточняться надлежащим образом.
- *Пользовательская история, или история,* — это описание функции или возможности с точки зрения пользователя продукта. Некоторые проводят различие между историей и пользовательской историей, но использование обоих понятий не привязано к стандарту — в большинстве случаев имеется в виду одно и то же.

Считается, что истории, как правило, нужно завершать за один спринт. Если в процессе уточнения будет обнаружено, что на историю требуется больше одного спринта, ее переквалифицируют в эпик.

- *Исправления*, *сокращение технического долга*, *костыли* — работа, ориентированная на разработку, которая не реализует пользовательские требования. Эти виды работ, как правило, называют «активаторами».

Терминология, которая ассоциируется с содержанием бэклога, избыточна и иногда неоднозначна. По этой причине некоторые эджайлисты просто говорят о «задачах бэклога продукта» или «элементах бэклога продукта» — эти понятия позволяют обойти множество проблем терминологии.

Как требования попадают в бэклог продукта

Бэклог играет ключевую роль в выполнении проектов по Agile, а многие руководства по Agile ничего не пишут о том, как владелец продукта и команда должны его заполнять.

Требования могут попасть в бэклог продукта посредством различных методов. В общем подходы можно описать как «сверху вниз» или «снизу вверх».

Выявление требований «сверху вниз»

При подходе «сверху вниз» процесс проработки требований начинается с общей картины. Команда определяет акторы, функционал, эпики, инициативы — высокоуровневые цели, функции и возможности бизнеса.

Затем их раскладывают на истории. Для внедрения подхода «сверху вниз» хорошо подходят следующие методы:

- Создайте карту историй.
- Определите видение продукта.
- Создайте презентацию для лифта.
- Напишите пресс-релиз и FAQ.
- Создайте план по шаблону lean canvas.
- Постройте карту влияния (impact map).
- Определите типы личности.

Каждый из этих методов предназначен для того, чтобы определить общее направление для релиза, и того, чтобы помочь создать более подробные истории, которые позволят выполнить реализацию.

Выявление требований «снизу вверх»

При подходе «снизу вверх» процесс проработки требований начинается с мелких подробностей, как правило, историй. Для внедрения подхода «снизу вверх» хорошо подходят следующие методы:

- Проведите пользовательскую историю — мастер-класс по написанию историй.

- Проведите фокус-группы с обычными пользователями.
- Проведите опросы по выявлению требований.
- Наблюдайте за тем, как пользователи выполняют свою работу.
- Проведите обзор отчетов о проблемах, полученных от текущей системы.
- Проведите обзор существующих требований (если воспроизводите функциональность существующей системы).
- Проведите обзор существующих предложений по улучшению.

После того как их создадут, истории объединяют в темы, функции и эпики.

«Сверху вниз» и «снизу вверх»

При разработке нового продукта обычно выбирается подход «сверху вниз». Подход «снизу вверх» хорошо подходит для унаследованных систем и эволюционной разработки. В новом проекте, в котором разработка ведется по итерациям, команда может сменить подход «сверху вниз» на «снизу вверх» после разработки достаточного количества функциональности, чтобы получить обратную связь от пользователей.

При использовании подхода «сверху вниз» трудности возникают из-за того, что подробности рассматриваются слишком поверхностно, чтобы увидеть весь объем работ, оставляют слишком много подробностей, чтоб позже при уточнении бэклога их прояснить.

При выявлении требований «снизу вверх» трудность состоит в том, чтобы получить ясный взгляд на всю систему целиком — то есть «лес не видно за деревьями». Вы можете не замечать ограничения верхнего уровня, которые важнее работы над мелочами. Необходима дополнительная работа для обеспечения того, чтобы работа команды в целом соответствовала согласованному направлению.

Подходы «снизу вверх» и «сверху вниз» в какой-то степени схожи в середине. Мастер-класс по написанию пользовательских историй может включать в себя множество методов «сверху вниз», набросок пресс-релиза можно использовать как подсказки в опросах по выявлению требований и так далее.

Подробности конкретных практических Agile-методов по выявлению требований в этой книге не приводятся. Раздел «Дополнительные ресурсы» в конце этой главы предлагает ресурсы, где можно найти больше информации.

Ключевой принцип: уточняйте бэклог продукта

После первоначального заполнения бэклога продукта его нужно постоянно уточнять, чтобы каждая задача бэклога содержала достаточно подробностей для эффективного планирования спринтов и разработки. Я всегда хочу видеть запас полностью уточненных задач бэклога примерно на два спринта, не считая работы в текущем спринте.

Недостаточное уточнение бэклога может вызвать много проблем у Agile-команды:

- Задачи бэклога не проясняются в достаточных подробностях, для того чтобы вести работу, поэтому команда идет в неверном направлении.
- Команда тратит слишком много времени на уточнение во время спринта и встречает слишком много неожиданностей в процессе работы.
- Задачи бэклога не обновлялись, поэтому команда реализует устаревшую концепцию продукта.
- Приоритет задач в бэклоге не ранжирован, поэтому команда работает над задачами, которые приносят меньшую ценность, и откладывает работу над более ценными.
- Задачи бэклога неправильно оценены и слишком крупны, поэтому команда не может уложиться в спринт — задача крупнее, чем ожидалось.
- Команда может откровенно изголодаться по работе, потому что в бэклоге недостаточно уточненных задач.

Сессии по уточнению бэклога

Уточнение бэклога осуществляется на сессии, на которой присутствуют владелец продукта, скрам-мастер и команда разработчиков. Присутствует вся команда, поэтому можно достигнуть общего понимания предстоящего объема работ.

Работа включает в себя обсуждение историй и эпиков, разбитие эпиков на истории, разбитие историй на более мелкие (и разбитие эпиков на более мелкие эпики), прояснение подробностей историй, определение критериев приемки для историй и оценки историй.

Встречи по уточнению бэклога обычно проводят в середине спринта. Если нужно ответить на вопросы, работу по подготовке ответа на них нужно провести до встречи по планированию следующего спринта, чтобы открытые вопросы не подрывали его планирование.

Владелец продукта приносит список задач, поставленных в порядке приоритета, для рассмотрения на сессии по уточнению бэклога, к тому времени он уже должен выполнить большинство работы, связанной с разработкой требований.

Ключевой принцип: создайте и применяйте критерии готовности задачи

Так же как критерии готовности помогают команде избежать перехода к следующему этапу работы до того, как работа на текущем этапе будет действительно выполнена, ясные и документированные критерии готовности задачи (DoR) помогают команде не начинать разработку по требованиям, до тех пор пока эти требования не будут готовы. Задача бэклога считается готовой при следующих условиях:

- Задача понята командой разработчиков достаточно хорошо, чтобы решить, можно ли ее выполнить в течение спринта.
- Задача оценена и ее можно выполнить за один спринт.
- Задача свободна от зависимостей, которые не позволят ее реализовать в течение спринта.
- Определены критерии приемки задачи (задача тестируема).

Команды могут создавать свои критерии готовности задачи, которые будут вариантами критериев, перечисленных выше. Цель состоит в том, чтобы необходимые задачи бэклога полностью уточняли до следующей встречи по планированию спринта, чтобы у команды были все сведения, нужные для эффективного планирования, и команда не отвлекалась на открытые вопросы.

Прочие соображения

Основы требований

Требования остаются острой проблемой долгое время. Agile поспособствовал развитию полезных практических методов для работы с требованиями, но важность высокого качества требований осталась неизменной.

При последовательной разработке проблемы с требованиями были заметны, потому что общая неэффективность была видна в конце проекта. В проекте, рассчитанном на год, если требования низкого качества принесли 10% потерь в эффективности проекта, проект завершится с опозданием более чем на месяц. Это проблема, которую невозможно игнорировать.

В Agile проблема от плохо определенных требований проявляется в меньших инкрементах чаще, по мере работы над проектом. Команда, потерявшая 10% в эффективности из-за плохо определенных требований, может переделывать историю каждые несколько спринтов. Это выглядит не так страшно, поскольку проблема не наваливается целиком, а накопленная неэффективность может быть значительна.

При проведении обзоров и ретроспектив Agile-команды должны быть особо внимательны к проблемам, связанным с требованиями. Команде, которая считает, что она недопонимает пользовательские истории, следует задуматься о целенаправленном развитии навыков работы с требованиями.

Обсуждение подробностей методов работы над требованиями в этой книге не предусмотрено, но вы можете оценить свои знания с помощью самопроверки, предложенной ниже. Если большинство терминов вам незнакомы, то стоит осознать, что в настоящее время требования к ПО — хорошо развитая дисциплина и существует много техник для работы с ними.

Самопроверка. Требования

Приемочное тестирование Разработка через приемочное тестирование (ATDD) Разработка через поведение (Behavior Driven Development) Контрольные списки (чек-листы) Контекстная схема Презентация для лифта Списки мероприятий Экстремальные персонажи Пять «почему?» Карта препятствий Оценка воздействия	Интервью Многоступенчатые вопросы Канвы (Lean canvas) Минимально жизнеспособный продукт (MVP) Персоны (Personas) Язык спецификации (Planguage) Пресс-релиз Видение продукта Прототипы Сценарии Карта историй Пользовательские истории

Рекомендации

Изучайте

- Проведите обзор подходов своих команд к требованиям через призму заблаговременности и своевременности. Как вы оцените «коэффициент порчи требований» (процент требований, которые устарели или которые нужно уточнить снова после того, как вы их определили, и до начала их реализации)?
- Для выявления требований ваши команды используют подход «сверху вниз» или «снизу вверх»? В какой мере, по вашему мнению, проявляются типичные для этих подходов трудности, описанные в этой главе? Есть ли у команд действующий план по их учету?
- Посетите сессию по уточнению бэклога, чтобы понять состояние бэклога команды. Достаточно ли определены требования для того, чтобы обеспечить эффективное планирование спринта и эффективность работы во время спринтов?
- Узнайте, есть ли у ваших команд критерии готовности задачи и применяют ли их.
- Просмотрите прошедшие обзоры спринтов и ретроспективы, определите задачи бэклога, которые невозможно было выполнить из-за недостаточного их уточнения. Что сделала команда, чтобы предотвратить такое в дальнейшем?

Приспосабливайтесь

- Примите меры для создания критериев готовности задачи.
- Примите меры для своевременного уточнения бэклога продукта.

Дополнительные ресурсы

[Карл Вигерс и Джой Битти, 2013]. Software Requirements, 3rd Ed. Это удобочитаемое, обширное описание практических методов для работы с требованиями как в Agile, так и в модели последовательной разработки.

[Сьюзан и Джеймс Робертсон, 2013]. Mastering the Requirements Process: Getting Requirements Right, 3rd Ed. Это хорошее дополнение к книге Вигерса и Битти.

[Майк Кон, 2004]. User Stories Applied: For Agile Software Development. Эта книга уделяет основное внимание данным пользовательских историй.

[Гойко Аджич и Дэвид Эванс, 2014]. Fifty Quick Ideas to Improve Your User Stories. Как можно понять из названия этой небольшой книги, в ней

представлено множество предложений по улучшению пользовательских историй.

ГЛАВА 14. Еще более эффективное определение приоритетов требований

Один из ключевых принципов Agile — выстраивание последовательности приоритетов доставки функционала от высшего к низшему, в зависимости от потребностей бизнеса. Истории с высшим приоритетом перемещают к верхней части бэклога для дополнительного уточнения и реализации в ближайших спринтах. Приоритеты используются также для того, чтобы решить, какие истории нужно реализовать, а какие нет.

Постановка приоритетов требований всегда приносила пользу, но при выполнении проектов по Agile на ней делается еще более заметный акцент. Для этого было разработано несколько действительно эффективных методов. Но сначала давайте рассмотрим роль, которая несет наибольшую ответственность за постановку приоритетов в Agile-проектах.

Владелец продукта

Как я уже объяснял в главе 4 «Начало эффективного Agile: Скрам», наиболее распространенной ошибкой в Скраме является назначение неэффективного владельца продукта. По опыту моей компании, эффективный владелец продукта обладает следующими характеристиками:

- *Компетенция в предметной области.*Эффективный владелец продукта отлично знает продукт, отрасль, а также пользователей продукта. Понимание отрасли дает основание для постановки приоритетов среди результатов команды, в том числе понимание того, что на самом деле нужно для создания минимально жизнеспособного продукта (minimum viable product, MVP). У него есть навык, необходимый для передачи позиции бизнеса команде технических специалистов.
- *Навык работы с требованиями*. Эффективный владелец продукта понимает уровень детализации и ее тип, необходимые для определения требований в соответствии со сложившимися обстоятельствами (уровень детализации требований к продуктам для бизнеса и медицинским приборам будет отличаться). Владелец продукта понимает разницу между требованиями и проектированием — он акцентирует внимание на вопросе «что?», а вопрос «как?» оставляет на усмотрение команды разработчиков.
- *Навыки фасилитации*. Хороший владелец продукта может привести людей к совместной цели. Работа с требованиями целиком заключается в согласовании противоречивых интересов: компромиссы между целями бизнеса и технических специалистов, компромиссы между локальными и более общими, организационными и архитектурными, техническими проблемами, устранение конфликта между различными заинтересованными сторонами и прочих напряжений. Эффективный владелец продукта будет помогать стейкхолдерам работать в разных направлениях, чтобы получить хороший продукт.

- *Смелость*. Время от времени эффективному владельцу продукта требуется способность принимать решения. Эффективный владелец продукта не тиран, но знает, когда решение требуется принимать именно ему, а не команде.
- *Общие характеристики личной эффективности*. Эффективный владелец продукта также обладает общими характеристиками эффективности — высоким уровнем энергичности, дальновидностью при уточнении бэклога, эффективностью проведения встреч и твердостью, чтобы доводить дело до конца.

Этот список желаемых характеристик подразумевает некоторый опыт, который нужен эффективному владельцу продукта. Идеальный владелец продукта обладает опытом в инжиниринге, продакшене, бизнесе, хотя, как я уже отмечал ранее, после соответствующей подготовки замечательный владелец продукта может получиться из бизнес-аналитика, сотрудника службы поддержки или тестировщика.

Размеры футболок

Как я писал в книге Software Estimation: Demystifying the Black Art [Макконнелл, 2006], оценка по размерам футболок — полезный способ определения приоритетов для частично уточненной функциональности, основанный на приблизительной окупаемости вложений.

При таком подходе технические специалисты классифицируют размер каждой истории (затраты на разработку) относительно других историй как малый (S), средний (M), большой (L) или очень большой (XL) (под «историями» можно понимать функции, требования, эпики и так далее). Помимо этого, такую же классификацию ценности для бизнеса используют клиенты, маркетологи, специалисты по продажам и другие стейкхолдеры, не относящиеся к техническим специалистам. Затем эти два набора данных сочетают, как показано в табл. 14.1 на следующей странице.

Таблица 14.1. Использование размеров футболок для классификации историй по ценности для бизнеса и затратам на разработку

История	Ценность для бизнеса	Затраты на разработку
История А	L (большая)	S (малые)
История Б	S (малая)	L (большие)
История В	L (большая)	L (большие)
История Г	M (средняя)	M (средние)
История Д	M (средняя)	L (большие)
История Е	L (большая)	M (средние)
...		
История ЯЯ	S (малая)	S (малые)

Такое соотношение ценности для бизнеса и затрат на разработку позволяет стейкхолдерам, не знакомым с технической стороной проекта, сказать что-то вроде: «Если затраты на разработку истории Бслишком большие, но ценность для бизнеса у нее малая, то она пока не нужна». Это

чрезвычайно полезное решение, которое можно принять на раннем этапе проработки такой истории.

Если бы вы провели некоторое уточнение истории, начали построение архитектуры, проектирование и так далее, вы бы потратили силы на историю, затраты на которую, в конечном счете, оказались бы неоправданными.

Быстрые ответы «нет» в отрасли разработки ПО обладают высокой ценностью. Размеры футболок позволяют принимать решения на ранних этапах проекта, чтобы отбраковать истории и не тратить на них силы.

Вопрос о том, что продолжить, а что сократить, легче решить, если отсортировать истории в приблизительном порядке по соотношению затрат/выгод.

Как правило, это делается путем присвоения значения, называемого «чистой ценностью для бизнеса», которое основано на сопоставлении затрат на разработку и ценности для бизнеса.

В табл. 14.2 показана одна из возможных схем вычисления чистой ценности для бизнеса в каждом из соотношений. Вы можете использовать эту схему или изобрести свою собственную, которая в вашем случае будет точнее отображать ценность, следующую из соотношения затрат на разработку и ценности для бизнеса.

К этой справочной таблице допускается добавление третьего столбца со значением приблизительной чистой ценности для бизнеса, по которому можно отсортировать истории, как в табл. 14.3.

Таблица 14.2. Приблизительная чистая ценность для бизнеса, основанная на соотношении затрат на разработку и ценности для бизнеса

Ценность для бизнеса	Затраты на разработку			
	XL	L	M	S
XL (очень большая)	1	5	6	7
L (большая)	−4	1	3	4
M (средняя)	−6	−2	1	2
S (малая)	−7	−3	−1	1

Таблица 14.3. Сортировка оцененных историй по приблизительной чистой ценности для бизнеса

История	Ценность для бизнеса	Затраты на разработку	Приблизительная чистая ценность для бизнеса
История А	L (большая)	S (малые)	4
История Е	L (большая)	M (средние)	3
История В	L (большая)	L (большие)	1
История Г	M (средняя)	M (средние)	1
История ЯЯ	S (малая)	S (малые)	1
История Д	M (средняя)	L (большие)	-2
...			
История Б	S (малая)	L (большие)	−3

«Приблизительная чистая ценность для бизнеса» — это именно приблизительное значение. Я не предлагаю просто вести обратный отсчет по списку и подвести черту. Польза от сортировки по приблизительной ценности для бизнеса заключается в том, что она помогает дать быстрый ответ «определенно да» касаемо историй вверху списка и «определенно нет» — в отношении историй внизу списка. Вам все равно понадобится обсудить задачи, находящиеся в середине. Поскольку чистая ценность для бизнеса — это приблизительное значение, иногда будут встречаться случаи, в которых при внимательном рассмотрении окажется, что история с чистой ценностью 1 важнее, чем история с ценностью 2.

Размеры футболок и единицы сложности историй

До этого в рассуждении об оценке историй по размерам футболок мы одинаково рассматривали размеры в качестве средства оценки как затрат на разработку, так и ценности для бизнеса. Если истории достаточно уточняли, для того чтобы оценить их в единицах сложности, то метод сработает одинаково хорошо, если затраты на разработку оценивались единицами сложности, но ценность для бизнеса по-прежнему оценивали по размерам футболок. Вы по-прежнему можете рассчитать приблизительную чистую ценность для бизнеса, чтобы наиболее рентабельные идеи попали в верх списка. Это можно обеспечить независимо от системы измерений при оценке затрат на разработку.

Карта историй

Поскольку бэклог продукта часто состоит из десятков или сотен историй, легко спутать приоритеты. Задачи, выполняемые к концу каждого спринта, могут быть несвязными, даже если по отдельности имеют наивысший приоритет в бэклоге.

Карта историй — мощная техника постановки приоритетов для выполнения историй в определенной последовательности и одновременного гармоничного распределения историй по пакетам [Паттон, 2014]. Она также помогает при выявлении, анализе и уточнении

требований, а еще — при отслеживании состояния проекта во время разработки.

Карта историй составляется всей командой и состоит из трех этапов:

1. Запишите основной функционал на стикерах и расположите их в порядке приоритета слева направо. Основной функционал состоит из функций, эпиков или крупных историй, тем, инициатив и прочих крупных требований. В оставшейся части этого обсуждения под эпиками я буду иметь в виду все вышеперечисленное.

2. Разложите эпики верхнего уровня на этапы и темы. Такая проработка не сместит приоритеты эпиков.

3. Разложите каждый этап или тему на истории и запишите их на стикерах. Разместите их под каждым этапом или темой в порядке приоритета.

В результате этих действий образуется карта историй, в которой перечислены требования в порядке приоритета слева направо и сверху вниз.

В следующих разделах эти этапы описаны более подробно.

Этап 1. Поставьте приоритеты эпиков и функционала верхнего уровня

Стикеры с функционалом верхнего уровня располагаются в порядке приоритета слева направо, как показано на рис. 14.1.

Приоритеты эпиков можно поставить с помощью размеров футболок или других методов, в том числе «Более ценная и короткая работа сначала», о котором рассказано в главе 22 «Еще более эффективное управление портфолио».

Эпики с низшим приоритетом с правой стороны карты историй могут быть не так важны, чтобы включать их в релиз. Если они годятся для релиза, они все еще могут быть недостаточно важны для включения в минимальный функционал продукта.

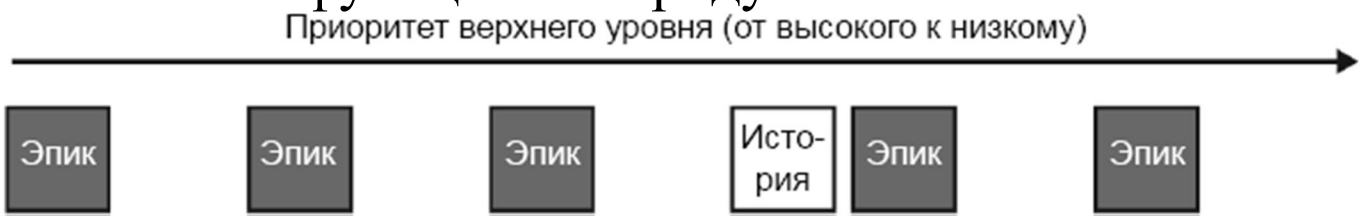


Рис. 14.1. Создание карты историй начинается с составления списка эпиков (и прочих задач верхнего уровня) слева направо в порядке приоритета

Этап 2. Разложите эпики верхнего уровня на этапы и темы

Большинство эпиков можно интуитивно описать как последовательность этапов. Некоторые, однако, могут не составлять такой последовательности, и их можно разложить на темы, как показано на рис. 14.2.

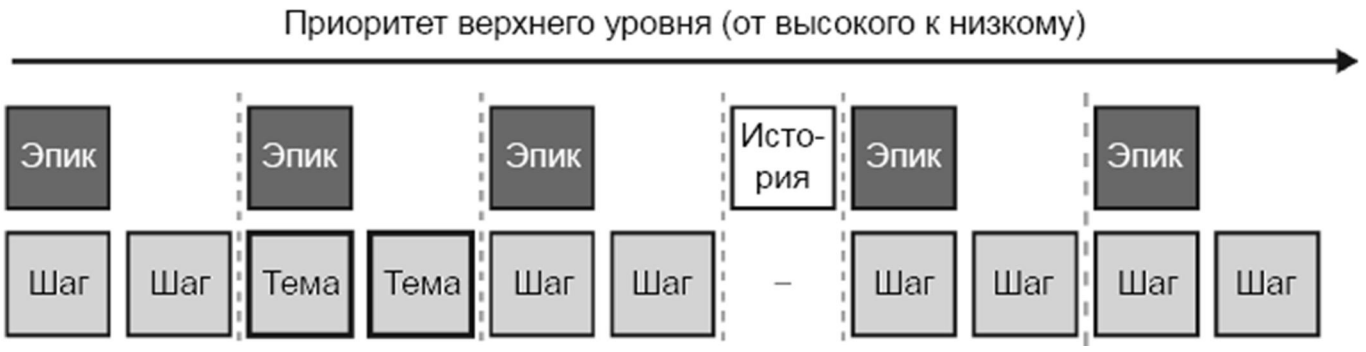


Рис. 14.2. На карте историй этапы или темы находятся под эпиками, что не меняет порядок приоритетов эпиков

В составлении карт истории этапы и темы, полученные после декомпозиции второго уровня, называются основной историей (хребтом, Backbone). Просмотрев описание основной истории, можно увидеть целостное описание общей функциональности, над которой ведется работа.

Этап 3. Разложите все этапы и темы на истории, расставляя приоритеты

Под полученной основной историей каждый этап или тема раскладывается дальше, после чего получается одна история или несколько. Они располагаются в порядке приоритета сверху вниз, как показано на рис. 14.3. Их сложность оценивается командой по размерам футболок или другим способом.

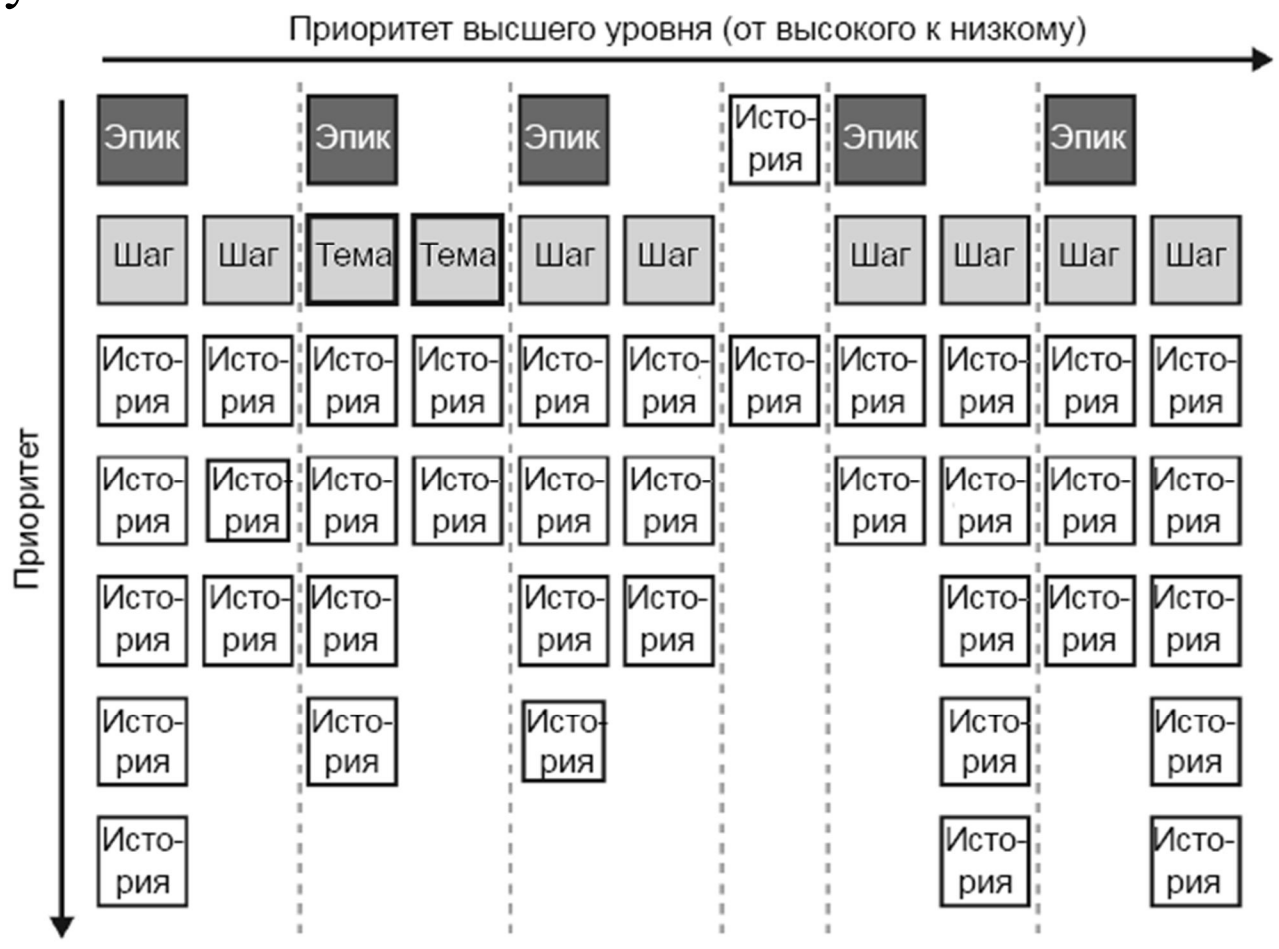


Рис. 14.3. Команда раскладывает каждый этап или тему на истории и сортирует их в порядке приоритета слева направо и сверху вниз

Вертикальные стеки историй под каждым этапом или темой называют «ребрами». Минимальный набор историй, находящийся непосредственно под основной историей, который составляет целостную реализацию, называется «ходячим скелетом». Ходячий скелет целостен, но обычно

недостаточен для того, чтобы стать MVP, в который входят дополнительные истории, находящиеся под скелетом.

Способ применения этих терминов к карте историй показан на рис. 14.4.

Команды могут составить длинные списки мелкой функциональности, которая не будет складываться в целостный релиз. Благодаря определению основной истории, ходячего скелета и MVP команда разработчиков способна ясно увидеть направление, в котором нужно продвигаться для доставки приоритетного и целостного функционала.

Карта историй и пользовательские роли

Как вариант, полезно начать с пользовательских ролей, находящихся вверху, а не с эпиков: расположить их в порядке приоритета слева направо, а затем разложить уже эпики под ними.

Карта историй как пример инфорадиатора

Эффективные Agile-практики делают акцент на том, чтобы работу можно было увидеть — сделать не только доступной на веб-странице, но и видимой частью рабочей среды. Карта историй на стене — это постоянное напоминание о приоритетах команд, текущих заданиях и дальнейшем рабочем процессе. Agile-команды называют такие средства отображения работы инфорадиатором.

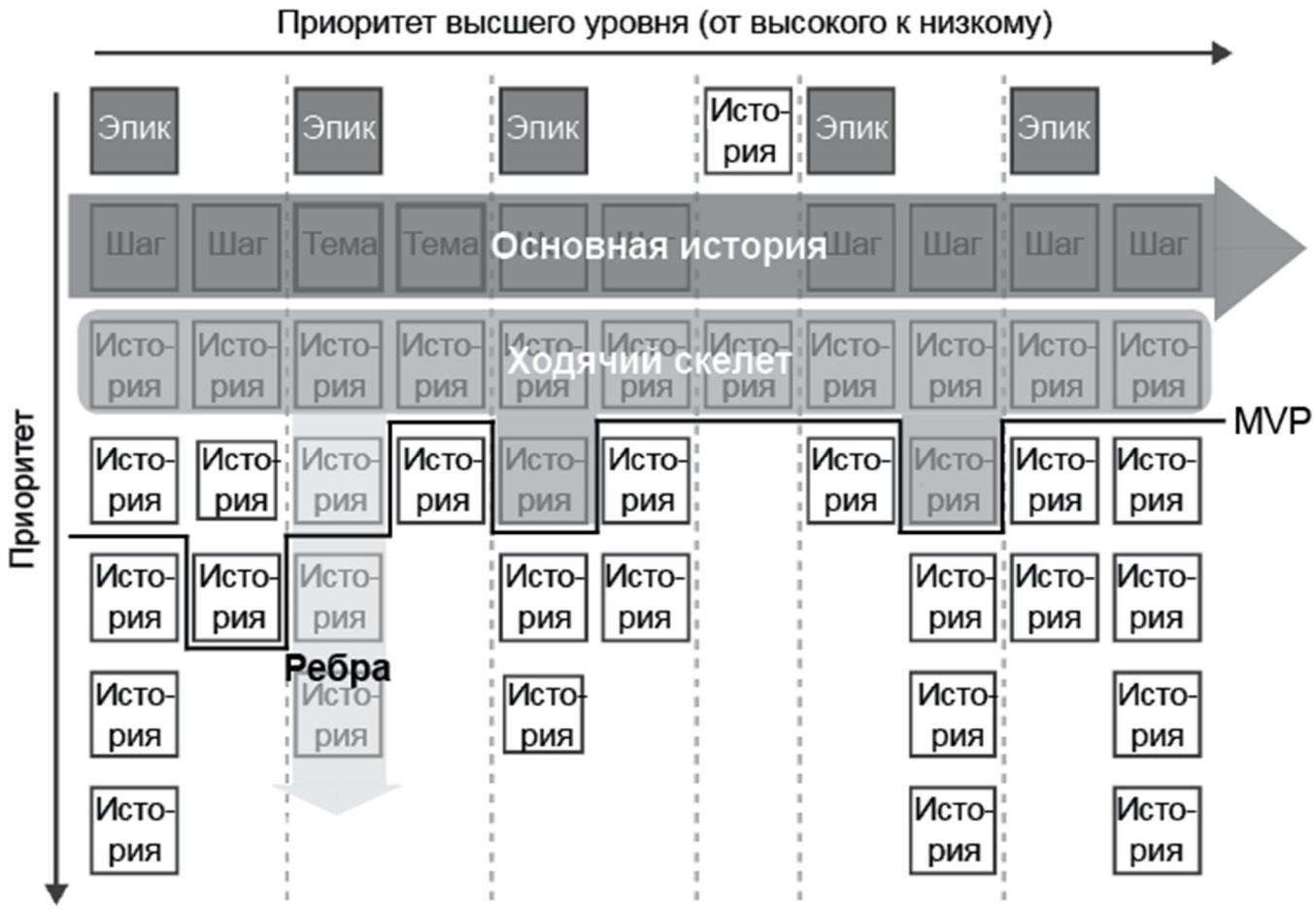


Рис. 14.4. Горизонтальный срез функциональности непосредственно под основной историей составляет минимальную целостную реализацию, называемую «ходячим скелетом». MVP обычно представляет собой нечто большее, чем ходячий скелет

Исследование показало, что наглядное отображение такого рода необходимо для улучшения производительности при доставке функционала [Форсгрен, 2018].

Карта историй как пример маятника Agile

Карта историй — увлекательный пример того, как маятник разработки ПО качнулся от чистой модели последовательной разработки до раннего Agile, а потом дошел до нынешней точки, когда Agile стал еще лучше. В самом начале при разработке по Agile специалисты любой ценой избегали какой-либо предварительной работы с требованиями — они прорабатывались только по мере необходимости. Карта историй, которая тесно связана с Agile, — это подход для организации и постановки приоритетов требований заблаговременно. Но это не тот старый метод полной проработки требований заранее, который использовался в последовательной разработке. Это метод, который помогает определить общие рамки для релиза заблаговременно, а затем способствует постановке приоритетов и ведет к пошаговому уточнению требований по мере хода работ.

Карта историй — это потрясающий пример того, как можно синтезировать последовательную разработку и Agile, чтобы получить лучшее из обоих направлений. Карта историй позволяет заблаговременно определять требования и уточнять их лишь незадолго до их реализации. Это помогает избежать распространенной в Agile ошибки, которая заключается в поставке функционала в порядке приоритета, но без представления общей картины. И прохождение через истории слева направо и сверху вниз часто выявляет недостающие этапы в эпиках, неправильное понимание приоритетов и прочие ошибки.

Прочие соображения

Как и в случае с выявлением требований, постановка их приоритетов долгие годы была острой проблемой. В дополнение к размерам футболок и карте историй рассмотрите следующие полезные методы.

Точечное голосование

При точечном голосовании у каждой из заинтересованных сторон есть ограниченное количество точек — например, 10. Заинтересованные стороны располагают свои точки среди требований так, как считают нужным. Все 10 точек на 1 требование, 1 точка на каждые 10 разных требований, 5 на одно и по одному на остальные — можно как угодно. Методика предлагает быстрый способ определить приоритеты группы.

Метод MoSCoW

MoSCoW — это сокращение от Must have, Should have, Could have, Won’t have, то есть «должны иметь», «желательно иметь», «могли иметь» и

«будем иметь». Это полезный метод распределения предлагаемых требований по категориям.

Минимально жизнеспособный эксперимент

Минимально жизнеспособный эксперимент (MVE) — это наименьший возможный релиз, с помощью которого можно получить ценную обратную связь для команды. MVE хорошо помогает при работе в контексте «Запутанные» по Cynefin, им, словно зондом, прощупывают возможное направление продукта.

Альтернативы минимально жизнеспособному продукту

Некоторые команды находят поддержание продукта на уровне минимума сложным. Если вы с командами сталкиваетесь с такими проблемами, рассмотрите альтернативные понятию «минимальный» формулировки. Например, «продукт ранней стадии тестирования», «продукт на ранней стадии использования» или «продукт на ранней стадии привлекательности».

Более ценная и короткая работа сначала

«Более ценная и короткая работа сначала» — это метод для максимального увеличения ценности, основанный на той последовательности, в которой эта работа выполняется. Об этом методе рассказано в главе 22.

Рекомендации

Изучайте

- Проведите анализ коллектива. Посмотрите, кто подходит на роль владельца продукта. Насколько эффективны специалисты, находящиеся в критически важных ролях? Они эффективные члены команды или слабые звенья цепи?
- Узнайте, какие методы применяют ваши команды для постановки приоритетов требований. Помогают ли эти методы выполнять реализацию в порядке, основанном на окупаемости?
- Узнайте, как ваши команды реализуют функциональность. Только на основе мелких подробностей и в порядке убывания ценности для бизнеса? Видят ли они общую картину?

Приспосабливайтесь

- Если ваши владельцы продукта неэффективны, подумайте об их переподготовке или замене.
- Попробуйте внедрить в командах метод оценки приоритетов задач бэклога продукта. Например, по размерам футболок или карте историй.
- Попробуйте реализовать в командах карту историй для создания целостных пакетов функциональности.

Дополнительные ресурсы

[Джефф Паттон, 2014]. User Story Mapping: Discover the Whole Story, Build the Right Product. Джефф Паттон обладает признанным авторитетом в области создания карт историй.

[Стив Макконнелл, 2006]. Software Estimation: Demystifying the Black Art. Раздел 12.4 этой книги содержит более подробное рассуждение об оценке историй по размерам футболок.

ГЛАВА 15. Еще более эффективная доставка функционала

Доставка — это деятельность, включающая в себя оставшуюся часть процесса разработки. Как таковой процесс доставки представляет собой линзу, которая позволяет рассмотреть более эффективные Agile-методы с нескольких сторон.

В этой главе я затрону темы как *доставки*, так и *развертывания*. Понятие «доставка» относится ко всячески необходимой для развертывания подготовке ПО, но без самого развертывания. Понятие «развертывание» относится к последним шагам, которые нужны для запуска продукта в производство.

Последний этап, необходимый для того, чтобы приступить к доставке, — интеграция. При разработке по Agile целью ставят как непрерывную интеграцию, так и непрерывную доставку или развертывание. Эти два метода лежат в основе DevOps.

Непрерывная интеграция не непрерывная в прямом значении этого слова. Под этим термином имеют в виду, что разработчики часто проверяют код в общем репозитории — обычно несколько раз в день. Точно так же непрерывная доставка не означает буквально «непрерывная». На практике это означает частую и автоматизированную доставку.

Ключевой принцип: автоматизируйте повторяющуюся деятельность

Деятельность при разработке ПО, как правило, разнится от свободной, творческой и ненормированной, такой как проработка требований и проектирование, до более ограниченной и нормированной, такой как автоматизированное тестирование, фиксирование изменений в транке (trunk), пользовательское приемочное тестирование (UAT), конвейеризация и продакшен. Люди хорошо справляются с более творческой первичной деятельностью, для которой требуется мышление, а компьютеры хороши в нормированных вторичных действиях, которые выполняются однотипно.

Чем ближе вы к доставке и развертыванию, тем больше смысла в автоматизировании деятельности и поручении ее компьютерам.

Некоторым компаниям лучше всего подойдет полностью автоматизированное развертывание, для которого нужен полностью автоматизированный конвейер развертывания, включающий автоматизацию однообразных задач. На рис. 15.1 показано, какие задачи потенциально автоматизируемы.

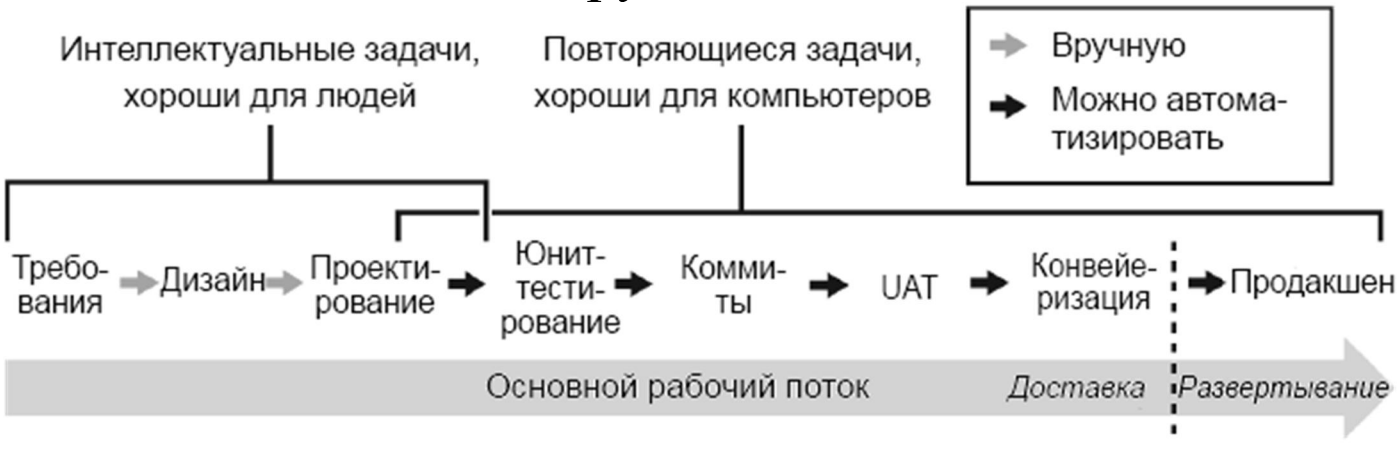


Рис. 15.1. Чем ближе работа к развертыванию, тем больше она подходит для автоматизации

Потенциал частоты развертывания по своей сути неограничен. Последние несколько лет *Amazon* развертывает системы каждые несколько секунд, более чем с тысячей развертываний в час [Дженкинс, 2011]. У большинства компаний нет обоснованных бизнесом причин, чтобы развертывать системы так часто, но производительность *Amazon* показывает, что возможна практически любая частота развертывания.

Как показано на рис. 15.1, ключевая идея в достижении автоматизированного развертывания — это отделение проработки требований, проектирования и конструирования, которые нельзя автоматизировать, от доставки и развертывания, которые можно.

Автоматизация на поздних этапах конвейера обеспечивает преимущества повышенной эффективности и более быстрого развертывания. Для человека это также полезно. Если рассматривать влияние автоматизации через призму автономии, профессионализма и целей, автоматизация еще и повышает мотивацию. Она позволяет избавиться от однообразных задач, при выполнении которых не остается возможности для роста, а также освобождает время, которое можно потратить на первичную деятельность, где просторы для роста есть.

Методы работы с поддержкой непрерывных интеграции и доставки

Для некоторых методов работы требуется поддержка непрерывных интеграции и доставки, о некоторых из них мы рассказали ранее.

Старайтесь автоматизировать все

Чтобы достичь полностью непрерывной интеграции или доставки, нужно полностью автоматизировать среду разработки. Автоматизация включает в себя управление версиями, которыми не получится управлять другим способом: версиями кода, конфигураций системы и приложений, сборок и конфигурационных скриптов.

Усильте акцент на автоматизации тестирования

Автоматизированная среда тестирования должна поддерживать каждое предложенное изменение, проходящее автоматизированные тесты нескольких различных типов, в том числе модульные тесты, тесты API, интеграционные тесты, тесты пользовательского интерфейса, тесты на случайные входные данные, тесты на случайные данные, нагрузочное тестирование и так далее.

Главное преимущество непрерывных интеграции и доставки заключается в автоматическом обнаружении и отказе регистрации изменений, которые недопустимы по причине внесения ошибок или ухудшения производительности.

Увеличьте приоритет развертываемости

Сопровождение автоматического конвейера развертывания требует усилий: для того чтобы непрерывные интеграция и доставка давали результат, команде нужно поставить приоритет поддержания системы в развертываемом состоянии над выполнением новой работы [Хамбл, 2015]. Приоритет новой работы над сопровождением автоматического конвейера разработки — хороший выбор, если вам нравится долго терпеть неудобства и продвигаться с низкой эффективностью.

Расширьте ваши критерии готовности

В то время как критерии готовности — это важная концепция в каждом проекте, их специфика становится еще важнее в среде непрерывных интеграции и доставки.

В такой среде нужно, чтобы критерии готовности включали в себя стандарты для юнит-тестирования, приемочного тестирования, регрессионных тестов, конвейеризации и управления редакциями. На рис. 15.2 изображены критерии готовности, которые подходят для такой среды.

- ☐ Все задачи бэклога продукта в инкременте продукта удовлетворяют приемочным критериям*
- ☐ Проходит статический анализ кода
- ☐ Проходит модульное тестирование без ошибок
- ☐ 70% функций с операторами проходят юнит-тестирование
- ☐ Завершено системное и интеграционное тестирование
- ☐ Все регрессионные тесты пройдены*
- ☐ ...
- ☐ Демонстрация в среде, похожей на производственную (конвейеризация)*
- ☐ Код находится в транке управления версиями, готов к релизу или в него выпущен*

Рис. 15.2. Пример критериев готовности для среды непрерывных интеграции или доставки. Критерии готовности, помеченные звездочками, отличаются от таких критериев для единственного проекта

Подчеркните дополнительные методы работы

Нужно предпринять несколько действий, для того чтобы добиться цели — минимизации временного интервала между появлением и обнаружением дефекта:

- Часто вносить код и изменения в него (хотя бы раз в день, а лучше чаще).
- Не фиксировать сломанный код.
- Устранять проблемы в конвейере развертывания сразу, в том числе нерабочие сборки.
- Писать автоматизированные тесты к коду.
- Обеспечить прохождение всех тестов.

Эти методы полезны в том, чтобы обеспечить состояние ПО на уровне релиза каждый раз, когда добавлена новая функция или внесено исправление.

Применяйте непрерывное развертывание как средство измерения общей эффективности разработки

Трата человеческих усилий на выполнение однообразных задач, которые могут выполнять компьютеры, — пустая трата сил. Время, требуемое на внесение изменений кода в производство, — полезное вспомогательное средство для оценки того, насколько ручной труд используется на конвейере развертывания.

Измерение времени на развертывание может стимулировать автоматизацию тестов, упрощение и автоматизацию процесса сборки, выпуска в релиз и развертывание, а также повысить внимание к проектированию приложений, не упуская из внимания его тестируемость и

развертываемость. Оно также может стимулировать разработку и развертывание функционала меньшими партиями.

Хамбл, Молески и О'Рейли дают такой совет: «Если чувствуете боль, вызывайте ее почаще, чтобы скорее перетерпеть» [Хамбл, 2015]. Другими словами, если что-то приносит вам боль, автоматизируйте, чтобы эту боль прекратить. Для вторичных действий, которые поддаются автоматизации, это отличный совет.

Преимущества непрерывных интеграции и доставки

Непрерывные интеграция и доставка приносят как очевидные преимущества, так и не очень. Очевидные преимущества — более быстрое и частое получение пользователями нового функционала. Неочевидные преимущества непрерывных интеграции и доставки могут быть еще важнее.

Команды быстрее учатся, потому что проходят через цикл «разработка — тестирование — релиз — развертывание» чаще, а это дает больше возможностей для обучения.

Недочеты обнаруживаются скорее после появления, соответственно их легче устранить, как об этом рассказано в главе 11 «Еще более эффективное качество».

Команды испытывают меньше стресса, поскольку автоматические релизы становятся легче — отсутствует человеческий фактор, который приводит к неудачному релизу.

Поскольку развертывание становится более надежным и обыденным, релизы можно производить в обычные рабочие часы. Если обнаружены ошибки, можно задействовать всю команду, а не вызывать только отдельных людей по звонку (причем еще и уставших).

Акцент на выпуске релизов несколько раз в день может принести преимущества, даже если вы работаете над критически важным ПО, которое редко выходит на суд пользователя. Частые релизы, даже внутри компании, помогают уделять качеству постоянное внимание. Они ускоряют обучение команды, поскольку каждый раз, когда релиз не получается завершить, у команды появляется возможность осознать причину провала и принять меры.

Наконец, как уже говорилось ранее в этой главе, непрерывные интеграция и доставка могут улучшить мотивацию тем, что у команд остается больше времени на работу, которая дает больше возможностей для роста.

Прочие соображения

Непрерывная доставка

Фраза «непрерывные интеграция и доставка» получила распространение в отрасли разработки ПО. Подразумевается, что для компаний непрерывные интеграция и доставка в порядке вещей. Однако в большинстве компаний непрерывная доставка отсутствует, есть только интеграция. Согласно исследованиям DZone, в 50% организаций были уверены, что они выполняют непрерывную доставку, и только 18% из них выполняли ее соответственно книжному определению [Исследование DZone, 2015].

Непрерывная интеграция является предпосылкой для непрерывной доставки, поэтому мы пришли к выводу, что есть смысл сначала добиться непрерывной интеграции. Несмотря на то что здесь много внимания уделяется компаниям вроде *Netflix* и *Amazon*, которые проводят развертывание сотни раз в день, организации, которые развертывают системы раз в неделю, месяц, квартал или реже, распространены в наши дни и будут в обозримом будущем. Может быть такое, что вы работаете над встроенными системами, над интегрированным в оборудование ПО, над регулируемыми системами, в корпоративном пространстве или над унаследованными системами, где частые релизы невозможны. Тем не менее вы все равно можете выиграть от автоматизации рутины при непрерывной интеграции. Вы также можете выиграть от дисциплины, которая связана с непрерывной доставкой, даже если непрерывное развертывание вам никогда не понадобится.

Это та область, где концепция границы Agile приносит пользу — у вас могут быть веские причины начертить эту границу так, чтобы она включала в себя непрерывную интеграцию, но без непрерывной доставки.

Граница Agile применима к внешним клиентам в той же мере, что и к разработке внутри организации. Нам доводилось работать с компаниями, которые развивали способность выпускать софт в релиз за пределы организации гораздо чаще, чем делают это на самом деле, — они выпускают релизы не так часто, потому что об этом их просят клиенты. Их клиенты находятся по другую сторону границы Agile. Они все равно часто выпускают внутренние релизы ради получения преимуществ, описанных в этой главе.

Рекомендации

Изучайте

- Изучите, насколько автоматизирован ваш конвейер доставки/развертывания.
- Опросите свои команды, чтобы определить, сколько усилий они затрачивают на однообразные действия, связанные с доставкой или развертыванием, которые можно автоматизировать.
- Составьте список действий во время процесса доставки/развертывания, которые все еще выполняются вручную. Какие действия мешают вашим командам производить доставку по нажатию кнопки?

- Узнайте, планируют ли ваши команды работу на уровне, позволяющем частую интеграцию.
- Задумайтесь об измерении времени от внесения изменений в код до развертывания.

Приспосабливайтесь

- Поощряйте людей интегрировать работу чаще, по меньшей мере раз в день.
- Создайте критерии готовности, которые поспособствуют автоматизации доставки и развертывания.
- Создайте для команд план по автоматизации сборок и сред разработки, насколько это возможно.
- Доведите до своих специалистов, что их работа по поддержанию конвейера разработки/развертывания в рабочем состоянии выше по приоритету, чем создание нового функционала.
- Поставьте цель сократить количество времени от внесения изменений в код до развертывания.

Дополнительные ресурсы

[Николь Форсгрен и др., 2018]. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. В этой книге изложены убедительные аргументы в пользу того, что конвейер разработки находится в центре внимания эффективной и здоровой организации доставки.

[Майкл Т. Найгард, 2018]. Release It!: Design and Deploy Production-Ready Software, 2nd Ed. В книге раскрыта тема проблем архитектуры и проектирования, а также проблем, связанных с более быстрым и надежным развертыванием.

ЧАСТЬ IV. Еще более эффективная организация

В этой части рассказано о проблемах Agile, которые лучше всего решать на уровне топ-менеджеров организации, а в некоторых случаях — *только* на уровне топ-менеджеров.

ГЛАВА 16. Еще более эффективное лидерство

Энтузиасты часто говорят о том, что реализации Agile зависят от принципа «лидерство как служение». Я считаю, что это так, но также считаю, что подобное определение слишком расплывчато, чтобы принести пользу во внедрении Agile. Требуются более прямые указания. Решили ли вы внедрить Agile в полном объеме или менее широко, лидерство — это решающая часть реализации Agile, поэтому в этой главе много ключевых принципов.

Ключевой принцип: управляйте результатами, а не мелочами

Компании живут и умирают в зависимости от принятых обязательств и их выполнения. Эффективные реализации Agile состоят из обязательств: команд и перед командами, руководства и перед руководством.

Agile-команды (в частности, работающие по Скраму) несут обязательства перед руководством, то есть к концу каждого спринта достигают своей цели.

При высококачественной реализации Скрама обязательства рассматриваются как абсолютные — команда будет работать чрезвычайно много, чтобы достичь целей спринта.

Взамен этого руководство обязуется перед командой, работающей по Скраму, оставлять спринт неприкосновенным. Руководство не станет изменять требования или как-либо еще вмешиваться в работу команды, находящейся в середине спринта. В проектах, выполняемых по модели последовательной разработки, это было неоправданным ожиданием, потому что циклы проекта были долгими и обстоятельства обязательно менялись. В скрам-проектах это полностью оправданно, потому что спринты, как правило, длятся 1–3 недели. Если организация не может сохранить фокус, не меняя за это время своих планов, то у нее еще бˆольшие проблемы, чем то, жизнеспособна ли ее реализация Скрама.

Идея восприятия команд и спринтов в виде черных ящиков и управления только входными ресурсами и результатами во время спринта обладает желательным побочным эффектом — помогает лидерам избежать микроменеджмента и укрепляет их положение.

Руководителям бизнеса нужно задать направление командам, объяснить цель работы, проработать приоритеты среди различных задач, а затем дать командам возможность удивить их результатами.

Ключевой принцип: выражайте цель с решимостью командира

Автономия и цель связаны, потому что команда не может обладать осмысленной здоровой автономией, если она не понимает цель своей работы. Самоуправляемой команде нужно принимать подавляющее большинство решений самостоятельно. У нее есть кросс-функциональные навыки и полномочия для этого. Однако если у нее нет ясно выраженного понимания цели своей работы, ее решения будут ошибочными (в буквальном смысле). На рис. 16.1 показаны возможные исходы для команды в зависимости от ее автономии и ясности поставленной цели.

Рис. 16.1. Автономия и ясность целей

Американские военные употребляют понятие «решимость командира», которое относится к публичному заявлению о желаемом конечном состоянии, целях операции и ключевых задачах, которые нужно завершить. Решимость командира особенно полезна в случаях, когда события разворачиваются не так, как планировалось изначально, коммуникации нарушены, а команде нужно принять решение без возможности передачи сведений выше.

Ваша цель в контексте разработки ПО схожа. Коммуникации с руководством компании, возможно, не будут принудительно нарушены, но часто бывает так, что к руководству компании проблематично обратиться в течение долгого времени [6], события разворачиваются не по изначальному плану, а команде все же нужно принимать решения. При таких обстоятельствах команды выигрывают благодаря «путеводной нити», или «полярной звезде», или «решительности командира», которые могут позволить им нащупать направление.

Правильное описание «решительности командира» будет включать следующее:

- Изложение причин и мотивации для проекта или инициативы; цель.
- Яркая визуализация желаемого конечного состояния. Это позволит членам команды понять, что нужно для достижения успеха и свое участие в этом.

Организации, желающей внедрить Agile, нужно развивать способности для четкого описания цели. Ее руководителям стоит сосредоточиться на достижении целей, а не на управлении с повышенным вниманием к мелочам.

Не говорите людям, как нужно делать. Скажите им, что нужно делать, и пусть они удивят вас своими результатами.

Джордж С. Паттон

Ставьте приоритеты и сообщайте о них

Эффективные руководители в Agile поддерживают свои команды тем, что сообщают однозначные приоритеты. Мы видели много организаций,

которые ставят высший приоритет на всем, поручая конечное решение командам. Они или меняют приоритеты слишком часто, или ставят приоритеты, слишком глубоко вдаваясь в мелочи, или вообще отказываются их ставить. Эти ошибки и широко распространены, и чрезвычайно снижают эффективность.

Отказ расставлять приоритеты — признак слабости руководства. Это равносильно отказу от ответственности за принятие решений. Если вам важно то, что будет сделано, вам придется принимать решения о приоритетах и недвусмысленно доносить их до сведения своих команд.

Частая смена приоритетов может оказаться столь же разрушительной. Она подрывает как чувство автономии команды, так и чувство цели. Руководителям стоит задать себе вопрос: «Будет ли такая смена приоритетов иметь значение через полгода?» Если нет, то это не так важно, чтобы дезориентировать команду.

Решительность командира — хорошая линза, через которую можно рассмотреть соответствующий уровень приоритетов. Руководителю нужно определить, что нужно для достижения успеха: цели, результаты, факторы и преимущества, но не углубляться в мелочи.

Расстановка приоритетов — это область, в которой эффективная реализация Agile может обозначить слабые стороны организации в формах, угрожающих ее лидерам. Нам встречались ситуации, когда руководители сворачивали реализацию Agile из-за того, что частая доставка (или ее нехватка) показывает неспособность лидера четко доносить приоритеты своим командам.

Важность этого нельзя недооценивать. Если вы неэффективно ставите приоритеты, то не руководите. Ваши проекты и близко не будут достигать результатов, которые потенциально возможны, а результаты будут далеки от тех, которых заслуживает ваша команда. Компания, желающая добиться эффективности, не будет избегать дискомфорта, возникающего от яркого света, проявившего слабые стороны при постановке приоритетов, — как раз она будет получать мотивацию к совершенствованию через этот дискомфорт.

Ключевой принцип: сосредоточьтесь на пропускной способности, а не на деятельности

Неэффективные руководители, как правило, больше обращают внимание на видимый, а не действительный прогресс. Но не каждое движение — это прогресс, а занятость часто является плохим показателем результативности.

Целью эффективной организации должно быть максимальное увеличение *пропускной способности* — скорости, с которой работа выполняется, а не скорости, с которой работа начинается, или уровня

активности. Руководители должны признать, что для максимального увеличения пропускной способности необходимо какое-то послабление [Демарко, 2002].

Одна из причин, по которой в Скраме подотчетность ведется на уровне команды, а не на индивидуальном, заключается в том, что это позволяет команде решить, каким образом удастся добиться наибольшей производительности. Если команда считает, что будет продуктивнее всего работать так, чтобы один из членов команды отдыхал целый день, то имеет полное право принять такое решение.

Давать людям какой-то запас времени — это хоть и нелогичный на первый взгляд способ увеличить пропускную способность, но, в конце концов, для компании важен общий результат каждой команды, а не результаты отдельных людей. Если производительность эффективно оптимизирована на уровне команды, компанию не должно беспокоить, что происходит на уровне отдельных специалистов.

Ключевой принцип: моделируйте ключевые стороны поведения при работе по Agile

Эффективные лидеры воплощают то поведение, которое они хотят видеть от специалистов, находящихся у них в подчинении. Такое поведение должно включать в себя:

- *Мировоззрение, ориентированное на рост,* — обязательство постоянно совершенствоваться как на уровне личности, так и на уровне организации.
- *Принцип «изучайте и приспосабливайтесь»* — постоянно размышляйте над своим опытом, учитесь, а затем применяйте усвоенное.
- *Декриминализация ошибок* — моделирование подхода, при котором каждая ошибка рассматривается как возможность для роста.
- *Корректировать систему, а не людей*— когда возникает проблема, воспринимайте ее как возможность для поиска изъянов в системе, а не вините специалистов.
- *Обязательство соответствия высокому качеству* — донесите своими действиями то, что поддержание высокого качества работ обязательно.
- *Фокус на бизнесе* — покажите, в какой мере соображения бизнеса наряду с техническими соображениями учитываются во время принятия решений.
- *Тесная обратная связь* — оперативно взаимодействуйте со своими командами (даже если им это не нужно, ведь ваша решимость командира дала ясное понимание направления работ).

Рекомендации

Изучайте

Пересмотрите свою производительность с точки зрения лидера:

- Воспринимаете ли вы свои команды как черные ящики? Что вас больше интересует — производительность и выполнение обязательств или управление мелочами?
- Ясно ли вы выражали свои намерения? С решимостью командира? Могут ли команды дать четкое и актуальное определение того, что такое успех в работе? Могут ли они работать несколько недель без вашего участия, если появится необходимость?
- Ставили ли вы четкие и реалистичные приоритеты для команд? Доносили ли вы эти приоритеты?
- На что вы больше обращаете внимание — на пропускную способность команды или на видимую деятельность?

Приспосабливайтесь

- Попросите команды провести полный анализ вашей деятельности как руководителя в соответствии с критерием «изучайте», приведенным выше. Поощряйте обратную связь от ваших команд, чтобы смоделировать обучение на ошибках.
- На основании результатов самооценки и входных данных команд разработайте список приоритетов по улучшению своих лидерских навыков.

Дополнительные ресурсы

[Штаб Корпуса морской пехоты США, 1989]. Warfighting: The U.S. Marine Corp Book of Strategy. Эта небольшая книга описывает подход Корпуса морской пехоты США к планированию и проведению операций. В этом описании я нашел много параллелей с ведением проектов по разработке ПО.

[Дональд Г. Рейнертсен, 2009]. The Principles of Product Development Flow: Second Generation Lean Product Development. В этой книге содержится распространенное рассуждение на тему пропускной способности, или «потока». Рейнертсен приводит убедительный аргумент в пользу того, что отсутствие внимания на потоке при разработке продукта, как он выражается, «неправильно до мозга костей».

[Том Демарко, 2002]. Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency. Демарко аргументирует в пользу того, что не нужно полностью загружать специалистов работой.

[Чад Сторли, 2010]. Manage Uncertainty with Commandet’s Intent. Harvard Business Review. 3 ноября 2010 г. В этой статье приводится немного более подробное описание «решительности командира», чем я привел в этой главе.

[Джон К. Максвелл, 2007]. The 21 Irrefutable Laws of Leadership. Книга Максвелла создает хороший контраст с чрезмерно аналитическим подходом к руководству, который я иногда вижу у руководителей в отрасли разработки ПО. Максвелл высказывает ключевые мнения,

например «сначала сердце, потом голова» и «люди не озадачены тем, сколько вы знаете, пока они не узнают, насколько вы озадачены».

ГЛАВА 17. Еще более эффективная культура организации

Большинство Agile-методов основаны на командном взаимодействии, они позволяют поддерживать в команде производительность, обучение и совершенствование. У руководителей также есть возможность расширять применение принципов команды до уровня организации. В этой главе описано, как способствовать применению Agile-методов на уровне организации.

Ключевой принцип: декриминализируйте ошибки

Как уже говорилось, ведение разработки по Agile основывается на цикле «изучайте и адаптируйтесь», который позволяет допускать ошибки, учиться на них и совершенствоваться. Под «допущением ошибок» я имею в виду принятие решения, когда вы понимаете, что не уверены в результате и готовы получить опыт независимо от того, каковым он будет.

С точки зрения Cynefin «сложные» проекты зависят от допущения небольшого количества допущенных ошибок, а «запутанные» — от большого количества. Поэтому очень важно, чтобы организация декриминализировала ошибки, чтобы они стали видимы, исследованы и в конечном счете выгодны организации, а не так, чтобы их прятали, стыдились, чем наносили ей вред.

Как говорит Джез Хамбл: «В сложной адаптивной системе сбои неизбежны. Когда происходят неполадки, человеческая ошибка — это начальная точка анализа, не стоит никого винить» [Хамбл, 2018].

Некоторые организации, например *Etsy*, предают ошибки гласности и их отмечают — идея, лежащая в основе этого, гласит: «Как хорошо, что мы сделали эту ошибку, потому что иначе мы бы никогда не узнали об этой проблеме».

Делайте нужные ошибки без промедления

В «запутанных» проектах важно не просто учиться на ошибках. В первую очередь важно непосредственно появление этих ошибок. Нужно создать такую культуру организации, где не возникнет опасения сделать ошибку, когда того требует необходимость. Как можно понять по рис. 17.1, это не дает права на ошибки по небрежности. Но полезно наладить культуру, при которой в случаях, когда исход решения нельзя предопределить заранее, команды не станут избегать возможности получить опыт и учиться на нем.

	Непреднамеренные	Допустимые (намеренные)
Медленные	Совершенствуйте навыки, допуская возможные ошибки, и учитесь на них	Делайте ошибки без промедления, чтобы скорее на них научиться
Быстрые	Избегайте ошибок из-за небрежности	Стремитесь больше всего ошибок допускать в этой категории, это необходимо для успешного решения проблем в контексте «Запутанные»

Рис. 17.1. Типы ошибок — таксономия декриминализированных ошибок

Устраняйте ошибки в допустимые сроки

Существуют допустимые сроки, в течение которых можно почти безболезненно решить проблему. Как только такой срок истекает, сложность решения проблемы начинает расти. Исправление ошибки во внутреннем релизе незатратно. Исправление ошибки после выпуска для клиентов уже затратно. Чем раньше всплыла проблема, тем выше шанс того, что ее удастся исправить, пока еще не вышли допустимые сроки. Хорошие новости распространяются быстро. Плохие новости нужно распространять еще быстрее.

Декриминализируйте эскалацию

Организации, которая серьезно настроена на декриминализацию ошибок, нужно декриминализировать и их эскалацию. Информация об ошибках должна беспрепятственно доходить до уровня, необходимого для исправления ошибки. *Microsoft* замечательно поработала над этим в начале 1990-х. Как-то раз мой босс пришел ко мне в кабинет и сказал: «Мне нужно выговориться. Я только что пришел со встречи с Биллом Гейтсом, вот мне там влетело! Я возился с проблемой две недели, а Билл заметил, что мог бы решить ее за пять минут, позвонив по телефону. Он отчитал меня за то, что я не сообщил ему. Мне паршиво, потому что я понимаю, что заслужил. Я ведь знал, что нужно было сообщить ему, а я не сообщил».

Психологическая безопасность

Декриминализация ошибок важна в том числе и потому, что позволяет обеспечить психологическую безопасность в команде. Двухлетний исследовательский проект, проведенный отделом персонала *Google*, выявил пять факторов, способствующих эффективности команды в *Google* (рис. 17.2).

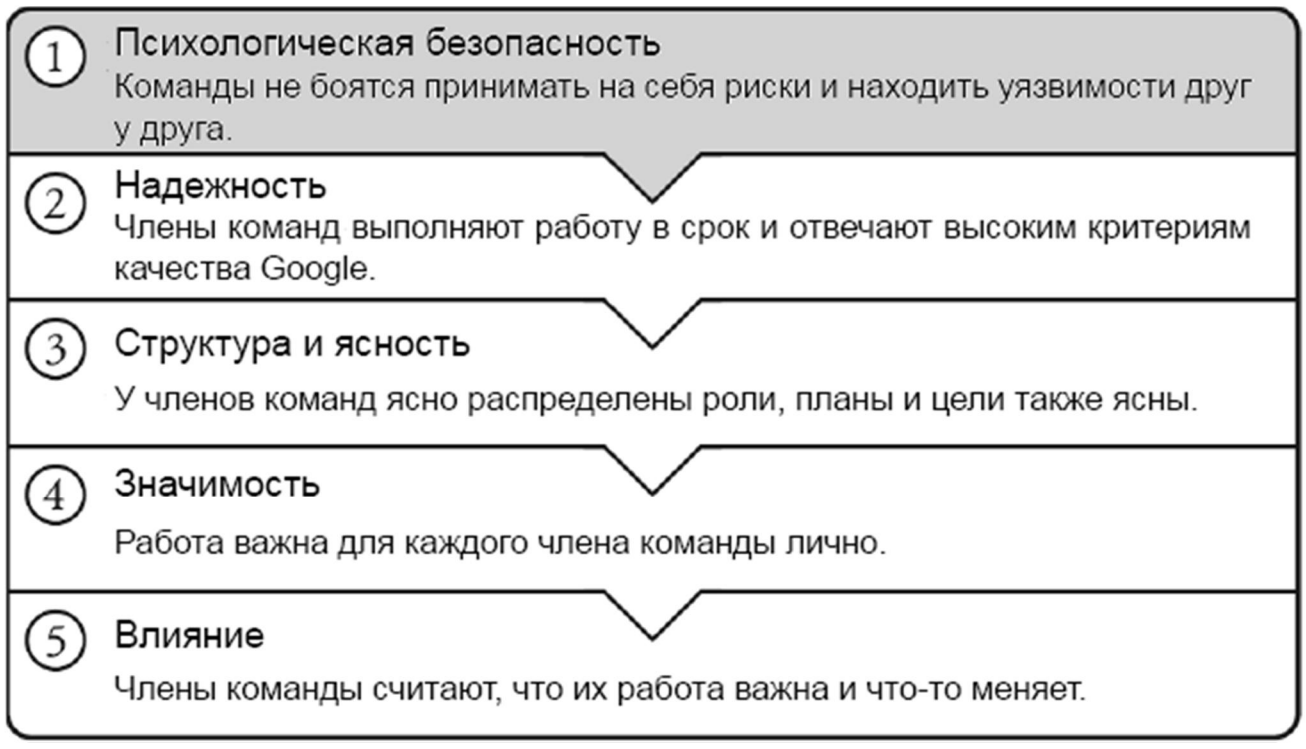


Рис. 17.2. Главный фактор успеха команды в Google — психологическая безопасность

Исследование *Google* обнаружило, что самым важным фактором, влияющим на эффективность команды, безусловно была психологическая безопасность, которой исследователи дают определение — возможность принятия рисков командой без чувства неуверенности или смущения. Они обнаружили, что:

Члены команд, в которых культивируется психологическая безопасность, с меньшей вероятностью покидают Google, чаще используют мощь различных идей, полученных от коллег, приносят больший доход, менеджеры в два раза чаще отмечают их эффективность [Розовский, 2015].

Исследование *Google* не противоречит исследованию, проведенному ранее Роном Веструмом [Веструм, 2005; Шу, 2001]. Веструм разработал «модель трех организационных культур», которая включает в себя патологическую, бюрократическую и производительную. Характерные свойства этих культур показаны в табл. 17.1.

Веструм обнаружил, что производительные культуры эффективнее, чем патологические и бюрократические: они превосходят ожидания, показывают большую гибкость и большую степень безопасности.

Таблица 17.1. Атрибуты разных культур в модели Веструма

Патологическая	Бюрократическая	Производительная
Ориентация на власть	Ориентация на правила	Ориентация на производительность
Слабое сотрудничество	Посредственное сотрудничество	Сильное сотрудничество
Наказание за плохие новости	Пренебрежение плохими новостями	Обучение тому, как доносить плохие новости
Уход от ответственности	Узкий круг ответственности	Риски, общие для всех
Препятствование связи между подразделениями	Допущение связи между подразделениями	Поощрение связи между подразделениями
Провал –> поиск виноватых	Провал –> суд	Провал –> запрос
Подавление нововведений	Проблемы из-за нововведений	Реализация нововведений

Для патологических организаций характерно подавление распространения плохих новостей. Производительные организации обнародуют плохие новости среди своих сотрудников. После провала они

делают запрос, благодаря чему плохие новости помогают совершенствоваться. Работа Веструма подтверждает важность декриминализации ошибок.

Ключевой принцип: стройте планы, основываясь на измеряемых возможностях команды

Эффективная организация рассматривает каждую команду и организацию в целом с точки зрения того, что у них есть определенные возможности для разработки ПО. Они зависят от производительности отдельных людей и всей команды, увеличения и уменьшения их численности, а также постепенного измеряемого улучшения производительности со временем.

Эффективная организация измеряет свои возможности и строит планы, основываясь на измеряемой и эмпирически подтверждаемой истории производительности, которая, как правило, основывается на скорости каждой команды. Этот подход контрастирует с другим, более интуитивным подходом, при котором компания создает планы, ожидая того, что ее команды покажут резкий скачок роста своих способностей (то есть чуда).

Разница в подходах к самооценке способностей к техническим работам заметна во время планирования портфолио проекта и установления сроков его выполнения. Если организация имеет ясное представление о своих способностях, она будет распределять работу и назначать сроки выполнения так, что команды будут с этим справляться. Если компания планирует, исходя из резких скачков роста способностей или не учитывая способности, она будет перегружать свои команды, что будет обрекать на неудачу как команды, так и организацию в целом.

Безапелляционные взгляды на возможности организации приводят к усилению давления на людей, дают почву для некоторых непреднамеренных и в конечном итоге разрушительных последствий:

- Команды не способны выполнять свои обязательства (цели спринта), что, в свою очередь, означает, что и компания не способна выполнять свои обязательства.
- Поскольку команды не могут справиться со своими обязательствами, у членов команд отсутствует чувство профессионального удовлетворения от проделанной работы, что подрывает их мотивацию.
- Чрезмерная нагрузка на команды противоречит мировоззрению, ориентированному на рост, а это мешает команде и организации совершенствоваться со временем.
- Чрезмерная нагрузка также приводит к выгоранию, увеличивает текучесть кадров и ухудшает возможности.

Как я уже писал более 20 лет назад в своей книге Rapid Development, руководители оказывают давление на свои команды в надежде на то, что такое давление создаст ощущение неотложности задач и подтолкнет к

правильной расстановке приоритетов. На деле попытка внушить неотложность задач чаще всего вызывает у команд полномасштабную панику, которая плохо сказывается на производительности, даже когда лидеры думают, что это давление отнюдь не велико [Макконнелл, 1996].

Сегодня Agile предлагает полезный набор инструментов для постановки приоритетов в работе как на уровне команды, так и на уровне организации. Лучше применяйте эти средства, чем оказывайте давление.

Учредите сообщества практиков

В организациях, с которыми нам доводилось работать, обнаружили, что учреждение сообществ практиков для распределения ролей по Agile улучшает производительность людей, выполняющих эти роли. Поскольку каждое такое сообщество состоит из людей, которые разделяют общие интересы в том, чем они занимаются, и в том, как делать это лучше, оно выбирает способ взаимодействия, который лучше всего подходит для его членов. Например, встречи могут проходить лично в реальном времени или по сети.

В сообществах практиков основное внимание может уделяться следующим обсуждениям:

- Общий обмен знаниями, коучинг младших членов сообщества.
- Обсуждение сценариев и решений распространенных проблем.
- Обмен опытом работы с инструментами.
- Обмен уроками, полученными из ретроспектив (и обратная связь).
- Определение областей организации со слабой производительностью.
- Нетворкинг.
- Выявление лучших практических методов внутри организации.
- Обмен мнениями по острым вопросам и взаимная поддержка.

Вы можете организовать сообщества практиков для скрам-мастеров, владельцев продукта, архитекторов, тестировщиков, консультантов по программе SAFe, Agile-коучей, специалистов по DevOps и т.д.

Участие в сообществе обычно является добровольным самостоятельным выбором, так что в нем находятся только те, кому там интересно.

Роль компании в поддержке еще более эффективного Agile

Некоторые атрибуты, необходимые для успешной работы команды, зависят от самой команды, но какие-то можно регулировать только на уровне организации.

Работая по Agile, команде невозможно добиться успеха, если компания препятствует ее усилиям. Сопротивление организации может выражаться в обвинении команд в допущенных ошибках, игнорировании потребности

команд в автономии, недостаточном донесении целей, а также в отказе дать командам возможность роста со временем. Конечно, такое возможно не только в Agile-командах, это может происходить с любыми командами.

Команды могут добиться наибольшего успеха, если компания поддерживает их тем, что создает культуру, в которой не принято обвинять, наполняет свои команды людьми, обладающими полным набором необходимых навыков, дает команде адекватную рабочую нагрузку, регулярно доносит цели, а также постоянно поддерживает профессиональный рост специалистов.

В зависимости от того, на каком этапе внедрения Agile вы находитесь, другим лидерам компании может понадобиться присоединиться к вам. Если вы обратитесь к границе Agile, описанной в главе 2 «В чем отличие Agile?», то можете определить тех самых лидеров и создать план того, как с ними работать.

Рекомендации

Изучайте

- Подумайте, как вы реагировали на ошибки, допущенные командой, последние несколько недель или месяцев. Как воспринимали команды вашу реакцию — как декриминализацию ошибок и возможность учиться на них? Вы моделируете свое поведение так, чтобы учиться на ошибках?
- Попросите членов команды оценить уровень своей психологической безопасности. Могут ли они принимать риски без чувства неуверенности или смущения?
- Проведите сравнительный анализ компании на соответствие производительной культуре по модели Веструма.
- Пересмотрите подход компании к распределению рабочей нагрузки на команды. Основаны ли ваши ожидания на эмпирически наблюдаемых способностях для выполнения работ?

Приспосабливайтесь

- Примите для себя решение декриминализировать ошибки при взаимодействии с командами.
- Донесите до команд, что вы ожидаете от них работы в устойчивом темпе, который позволяет учиться и расти. Попросите их сообщить вам, если график работ противоречит таким ожиданиям.
- Создайте план по устранению недочетов, выявленных при сравнительном анализе с использованием модели трех организационных культур Веструма.
- Составьте план того, как привлечь других лидеров компании к участию в реализации Agile.

Дополнительные ресурсы

[Джулия Розовски, 2015]. The five keys to a successful Google team. Сетевой ресурс (17 ноября 2015 года). (Дата доступа: 25 ноября 2018 г.) https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/. Эта статья описывает работу над культурой организации в Google.

[Рон Веструм, 2005]. A Typology of Organisational Cultures. Журнал Quality and Safety in Health Care. Январь 2005, с. 22–27. Это исчерпывающая работа Веструма по модели трех организационных культур.

[Николь Форсгрен и др., 2018]. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. В этой книге содержится рассуждение о модели организационных культур Веструма применительно к организациям в области ИТ.

[Билл Кертис и др., 2009]. People Capability Maturity Model (PCMM), Version 2.0, 2nd Ed. В этом документе описан подход к совершенствованию практических методов по управлению кадрами в технических организациях. Подход логичен, а значение представляется очевидным. Документ может быть труден для чтения. Я предлагаю начать с рис. 3.1 для понимания контекста.

ГЛАВА 18. Еще более эффективные измерения

При менее эффективных реализациях Agile измерение иногда воспринимается как враг. При более эффективных реализациях Agile измерения используются для того, чтобы принимать решения об изменении процесса на основе количественных данных, а не только на основе субъективного мнения.

Это первая из трех глав, в которых рассказано о количественном подходе при разработке по Agile. В этой главе описано, как создать эталон измерений, имеющий смысл. В главе 19 «Еще более эффективное совершенствование процесса» рассказано, как использовать измерения для улучшения процесса и производительности. В главе 20 «Еще более эффективная предсказуемость» рассказывается об оценке работ.

Измерение количества работ

Измерение начинается с измерения объема работ. В Agile-проектах измерение работы ведется в единицах сложности. Единица сложности — это единица измерения размера и сложности задачи. Команды, работающие по Agile, используют единицы сложности историй для оценки, планирования и отслеживания своей работы. Единицы сложности историй полезны для измерения улучшений не только процесса, но и производительности.

Agile-команды чаще всего используют шкалы единиц сложности историй на основе чисел Фибоначчи от 1 до 13 (1, 2, 3, 5, 8 и 13). Размер каждой рабочей задачи оценивается в единицах сложности историй, затем размеры задач складываются, чтобы в сумме был размер всей работы в единицах сложности.

Числа, не входящие в последовательность Фибоначчи, к примеру 4 и 6, не используются. Это позволяет избежать ложной точности, когда спорят, какой сложности история, например 5 или 6, хотя команда на самом деле не знает, оценивать ли историю в 3, 5 или 8 единиц сложности.

В совершенном мире существовал бы универсальный эталон измерения и присвоения единиц сложности. Но в реальности каждая команда самостоятельно определяет шкалу, насколько велика единица сложности именно для этой команды. Проработав со шкалой некоторое время, команда приходит к единому мнению, сколько означает 1, сколько означает 5 и так далее. Большинству команд нужно получить некоторый опыт в присвоении историям единиц сложности, прежде чем шкала единиц историй стабилизируется.

После того как единицы сложности были присвоены, команда не меняет оценки историй в единицах сложности, основываясь на своей фактической производительности. Если изначально историю оценили в 5 единиц сложности, но ко времени выполнения кажется, что этих единиц должно быть не меньше 8, история по-прежнему оценивается в 5 единиц.

Скорость

Когда объем работы уже установлен и измерен в единицах сложности историй, следующим этапом будет вычисление скорости, с которой продвигается работа.

Скорость команды представляет собой определенное количество единиц сложности, выполняемое за спринт. Если команда успела выполнить 42 единицы сложности историй в этом спринте, то ее скорость составляет 42 единицы сложности в спринт. Если команда за один спринт выполняет 42 единицы сложности, в следующем — 54, потом 51, а потом 53, то в итоге средняя скорость команды составит 50 единиц сложности в спринт.

В отдельных спринтах скорость команды колеблется и, как правило, незначительно. Более важны показатели средней скорости со временем. Как только команда установила эталонную скорость, которая, как она считает, точно отражает темп хода работ, команде можно начать экспериментировать с изменением процесса и наблюдать, как эти изменения влияют на скорость команды. Подробнее о том, как это сделать, рассказано в главе 19.

Некоторые команды также отслеживают скорость роста объема работ, которая показывает, с какой скоростью в текущий проект добавляется новая работа.

Небольшие истории

Для общего применения (не для измерения показателей) некоторые команды используют дополнительные значения единиц сложности

историй, например 21, 40 и 100 (круглые числа) или 21, 34, 55, 89 (числа Фибоначчи), чтобы оценивать темы, эпики и прочие более крупные задачи бэклога.

Чтобы вести измерения, которые имеют смысл, истории нужно раскладывать так, чтобы их оценка укладывалась в значение от 1 до 13, причем командам необходимо позаботиться о том, чтобы распределить единицы сложности историй пропорционально. История, оцененная в 5 единиц, должна относиться к истории, оцененной в 3 единицы, как 5/3. Это позволяет команде выполнять важные числовые операции, такие как сложение единиц сложности историй и получение их итогового количества.

Числа вроде 21, 40 и 100 для такого не подходят. Они скорее представляют собой метафору, в целях получения измерений их применять не стоит.

Короткие итерации

Скорость рассчитывается на основании результатов спринта, поэтому чем короче будут спринты, тем чаще вы сможете обновлять данные о скорости команды. По сравнению с последовательной разработкой, в которой итерации занимали весь жизненный цикл и длились целые кварталы или даже годы, в Agile короткие итерации позволяют откалибровать скорость команды лишь за пару месяцев.

Сравнение скорости команд

Каждая команда сама себе создает шкалу единиц сложности историй, основываясь на специфике выполняемой работы. Лидеры, естественно, хотят иметь возможность сравнивать производительность команд, но у разных команд характер работ может сильно отличаться, чтобы можно было провести полноценное сравнение. Команды могут различаться по следующим параметрам:

- Разные виды работ (новая и унаследованная система, бэкенд и фронтенд, наука и бизнес и так далее).
- Разные технические стеки или разные части одного стека.
- Разные стейкхолдеры, которые оказывают разный уровень поддержки.
- Разное количество специалистов в командах, включая специалистов, которых добавляют и отстраняют в разное время.
- Разная ответственность за поддержку продукта.
- Разные потери в обычной скорости из-за обучения, отпусков, релизов, местных праздников и т.п.

Даже несмотря на то, что все команды используют единицы сложности историй, сравнить скорость разных команд полноценно не получится. Это как если бы одна команда играла в бейсбол, другая — в футбол, а третья —

в баскетбол. Или если одни играют в NBA, а другие — в NHL. Сравнивать забеги, цели и единицы сложности у разных команд не имеет смысла.

Руководители, которые попытались сравнивать производительность команд с помощью скорости, говорят, что ни к чему хорошему это не приводило. Такое сравнение стравливало команды между собой. Команды осознают, что сравнения построены на ненадежных данных, поэтому воспринимают такие сравнения как несправедливые. В результате снижаются моральный дух и производительность — происходит обратное тому, чего ожидает руководство первым делом, сравнивая команды.

Измерение качества работ

Помимо количества работ можно измерить и их качество. Его даже нужно измерять, чтобы команда, обращая внимание на количество, не забывала о качестве.

Процент доработки — это процент усилий, затрачиваемых на доработку недочетов относительно выполнения новой работы. Как я уже упоминал в главе 11 «Еще более эффективное качество», доработка при выполнении проекта по разработке ПО — полезный показатель низкой эффективности или расточительного расхода усилий. Высокий процент доработки может указывать на то, что команда недостаточно времени тратит на уточнение историй до их реализации, что у нее отсутствуют достаточно строгие критерии готовности или она их не придерживается, не проводит должного тестирования, позволяет себе накапливать технический долг, а также на другие проблемы.

В проектах, выполняемых по модели последовательной разработки, материал для доработки, как правило, накапливается к концу проекта, причем в планах нет никакой доработки, поэтому она вполне заметна. В командах, работающих по Agile, доработка, как правило, выполняется постепенно, поэтому ее труднее заметить. Но она все равно есть, поэтому при работе по Agile полезно отслеживать процент доработки в командах.

Применение единиц сложности представляет собой основу для измерения доработки. Истории можно классифицировать как новую работу или доработку. Процент доработки рассчитывается как количество доработки в единицах сложности, деленное на общее количество работ в единицах сложности историй. Команда со временем может отслеживать изменение процента доработки в бˆольшую или меньшую сторону.

Командам нужно приходить к консенсусу относительно доработок. Командам, работающим над унаследованными системами, следует считать доработку проблем, созданных другими командами ранее, новой работой. Командам, которые исправляют свои ошибки, сделанные ранее, следует считать такую деятельность доработкой.

Процент доработки можно измерить альтернативным способом, просто-напросто не присваивая доработке единиц сложности историй. У вас не получится вычислить коэффициент доработки, но если команда тратит на доработку много времени, вы увидите, что скорость команды падает, потому что время, потраченное на доработку, не приносит команде выполнения единиц историй.

При любом подходе цель — найти равновесие между количественным измерением скорости и измерением качества работ.

Общие соображения при измерениях

При использовании средств измерения, характерных для Agile, руководителям команд нужно иметь в виду несколько ключевых факторов, позволяющих успешно провести измерения при разработке ПО.

Расскажите о своих ожиданиях от измерений

Открыто говорите о том, зачем вы проводите измерения и как вы собираетесь применять их результаты. Команды разработчиков беспокоятся о том, что измерения могут использовать нечестно или неправильно, при том что репутация многих компаний создает повод для такого беспокойства. Дайте ясно понять, что измерения направлены на поддержку саморазвития команд — это поможет внедрить проведение измерений.

То, что измеряемо, выполняется

Если вы только измеряете что-то одно, люди естественным образом оптимизируют свою работу под такие измерения, а это может привести к нежелательным последствиям. Если вы измеряете только скорость, команды могут урезать ретроспективы, пропускать ежедневные митинги, ослаблять критерии готовности и накапливать технический долг, лишь бы выполнять работу быстрее.

Убедитесь в том, что вы задействуете набор мер, необходимых для оптимизации деятельности команд, в том числе обеспечиваете приемлемый уровень качества и удовлетворения клиентов, так чтобы команды не оптимизировали свою работу ради скорости в ущерб другим задачам, которые не менее важны.

Схожим образом нужно измерять то, что наиболее важно, а не только то, что измерить легче всего. Если бы у вас была команда, которая доставляет лишь половину единиц сложности историй от необходимого, но приносит вдвое больше ценности для бизнеса, это было бы слишком просто, согласитесь? Поэтому будьте уверены, что измерение единиц сложности историй не приводит к неумышленному отвлечению внимания вашей команды от доставки ценности для бизнеса.

Прочие соображения

Используйте данные, полученные от инструментов, с осторожностью

Компании вкладываются в инструменты, а технические специалисты вводят данные о дефектах, данные учета времени и данные пользовательских историй. Компания, естественно, полагает, что данные, собранные с помощью инструментов, верны. Но зачастую это не так.

Мы работали с одной компанией, которая была уверена, что у нее были точные данные учета времени, потому что в течение многих лет руководство требовало, чтобы сотрудники отмечали начало и окончание рабочего дня. Когда мы проанализировали данные, то нашли множество аномалий. Затраты времени на два проекта, на которые должно быть затрачено одинаковое количество усилий, различались в сто раз. Мы выяснили, что люди не понимали, зачем был нужен сбор данных, и считали, что это бюрократия. Один сотрудник написал скрипт, который автоматически вводил данные учета времени. Этот скрипт получил широкое применение, неизменно у каждого вводились одни и те данные! Другие члены коллектива вообще не вводили никаких данных учета времени. И данные в итоге не несли никакого смысла.

Рекомендации

Изучайте

- Поговорите со своими командами на тему их отношения к измерениям. Понимают ли команды, что измерения помогают им вносить изменения в рабочий процесс, которые в конечном итоге улучшают качество условий работы?
- Просмотрите размеры историй и длительность итераций в командах. Достаточно ли малы размеры историй и длительность итераций, чтобы обеспечить более точные измерения производительности?
- Каким образом вы измеряете качество? Находится ли ваш способ измерения качества в равновесии со средствами измерения количества? Другими словами, учитывает ли ваш способ все измерения, важные для вашего бизнеса?
- Просмотрите любые данные, собираемые с помощью инструментов, которые использует ваша организация. Изучите, соответствуют ли полученные данные тому, что вы о них думаете.

Приспосабливайтесь

- Донесите командам, что цель проведения измерений — способствовать их работе.
- Поощряйте применение вашими командами единиц сложности историй и единиц измерения скорости, если они их еще не применяют.
- Поощряйте использование командами способов измерения качества, например измерения процента доработки, если такие способы еще не применяются.

- Прекратите использование измерений, которые не несут смысловой нагрузки или вводят в заблуждение, в том числе использование невалидных данных, а также результатов некорректного сравнения производительности различных команд.
- Предупредите вашу организацию об опасности сравнения скорости разных команд, если требуется.

Дополнительные ресурсы

[Джон Белбуте, 2019]. Continuous Improvement in the Age of Agile Development. Эта практическая книга содержит развернутое рассуждение об измерениях деятельности команд разработчиков, а также об улучшении процесса разработки, в основном с упором на качество.

ГЛАВА 19. Еще более эффективное совершенствование процесса

Как бы вы описали эффективный подход к Agile в нескольких словах? Мой ответ таков: «Меняйте систему, а не людей». О декриминализации ошибок и о ее важности я уже говорил.

Но декриминализация ошибок не означает их игнорирования — это означает, что нужно сообща, открыто, со взаимным уважением помогать друг другу понимать факторы, которые привели к ошибке, и изменить их так, чтобы схожая ошибка не повторилась снова.

Есть распространенное убеждение, что Agile — это «работать настолько быстро, насколько это возможно», но такая установка будет мешать подлинному совершенствованию. Более эффективные реализации Agile сосредоточены на том, чтобы достигать большей скорости за счет улучшения эффективности.

Скрам как эталон для улучшения процесса

Если мы вернемся в дни расцвета модели зрелости возможностей (Software Capability Maturity Model, сокращенно SW-CMM), то уровень 2 будет представлять собой «повторяемый» процесс. Тем самым был установлен эталон, который позволял проводить измеряемые улучшения на более высоких уровнях модели зрелости возможностей. Высококачественная реализация Скрама служит той же цели. У скрам-команды есть эталонный процесс, которому она неизменно следует и который служит фундаментом для дальнейших улучшений.

Повышение производительности

Команда горит желанием улучшить производительность, но уверены ли вы, что ваша команда совершенствуется? Как вы измеряете производительность?

Хотя производительность в разработке ПО практически невозможно измерить в абсолютных единицах, единицы сложности историй и скорость команды позволяют провести относительные измерения, благодаря чему удается измерить изменения в производительности, что дает плодородную почву для резкого улучшения производительности команды.

Сравнение успехов команды с течением времени — это первостепенное, правильное применение единиц сложности историй для измерения производительности. Если команда в среднем достигала 50 единиц первые пять спринтов, а вторые пять спринтов скорость в среднем достигала 55 единиц, то это говорит о том, что производительность команды возросла.

Улучшение производительности команд

Первый этап улучшения производительности — это создание измеряемого, правдоподобного эталона на основе измерений скорости, как описано в главе 18 «Еще более эффективные измерения».

Когда вы установили эталон скорости, то вн^осите изменения и сравниваете эталонную скорость команды с ее скоростью в течение следующих нескольких спринтов. Через несколько спринтов вы получите представление о том, как увеличилась или уменьшилась производительность.

Приведем несколько примеров изменений, последствия которых можно измерить, вычислив изменения скорости команды со временем:

- Внедрение нового инструмента сотрудничества.
- Изменение части технологического стека.
- Перевод владельца продукта в географически распределенной команде из штата на удаленную работу.
- Ужесточение критериев готовности и увеличение временн^ых затрат на уточнение историй перед началом работ по реализации.
- Изменение длительности каденций от 3 недель к 2.
- Перемещение команды в опен-спейс из закрытой площадки.
- Добавление нового члена команды в процессе релиза.

Конечно, новые цифры не всегда будут окончательными. В общем, при измерении производительности полезно принять, что измерения наталкивают на некоторые вопросы и размышления, но не обязательно дают необходимые ответы.

Чего возможно достичь благодаря улучшению производительности команды

Короткие спринты часто открывают возможности для экспериментов с изменением процесса, отслеживания результатов изменений и достижения успеха на основе этих изменений. С таким подходом улучшения накапливаются быстро. Нам встречались команды, которые увеличивали производительность в два раза или больше.

Результаты могут быть неожиданными. Нам неоднократно встречались случаи, когда команда хотела «выбрать голосованием проблемного члена команды и отстранить его от работы» из-за низкой производительности. В каждом случае история была схожей. Менеджер спрашивал: «Если мы отстраним этого человека, вы сможете поддерживать вашу скорость на том же уровне?» Команда отвечала: «Да, мы станем работать еще быстрее, потому что он тянет нас вниз».

В другом случае мы работали с фирмой, занимавшейся цифровым контентом, у которой команды работали на двух разных площадках. На первой площадке работала команда из 15 специалистов, а на второй — из 45. Благодаря тщательному отслеживанию скорости, контролю выполняемой работы и анализу состояний ожидания команда на первой площадке решила, что больше времени и усилий тратит на координацию работ со второй площадкой, чем получает от нее пользы. Второй площадке отменили проект и поручили другой, а общая отдача от изначального проекта возросла, хотя над ним продолжили трудиться только 15 специалистов на первой площадке. Они эффективно увеличили свою производительность благодаря использованию средств измерения производительности.

Влияние компании на производительность

Знатоки Скрама не перестают повторять мантру о том, что «Скрам не решает ваши проблемы, но он проливает яркий свет на них, и вы можете понять, что они собой представляют». Иногда Скрам обнажает проблемы, которые команда может решить самостоятельно, а иногда — те, которые нужно решать на уровне организации. Часто встречающиеся проблемы, связанные с организацией, могут быть такими:

- Сложность принятия на работу квалифицированных специалистов.
- Высокая текучесть кадров.
- Слабое профессиональное развитие.
- Слабая подготовка руководителей.
- Неготовность к отстранению проблемных членов команды.
- Неготовность следования правилам Скрама, например внесение изменений в середине спринта.

- Неспособность наполнения особых ролей (скрам-мастер, владелец продукта).
- Частые изменения в направлении бизнеса.
- Зависимость от других команд, неоперативность других команд.
- Избыточная многозадачность при ведении нескольких проектов, в том числе при требуемой поддержке продукта.
- Недостаточная поддержка от бизнеса, низкая скорость принятия решений.
- Медленное принятие решений исполнительным руководством.
- Бюрократические методы работы компании.
- Команда разбросана по нескольким площадкам.
- Недостаточная поддержка перемещения между площадками.

Сравнение производительности различных команд

Хотя провести полноценное сравнение скорости различных команд невозможно, есть один показатель, по которому сравнивать есть смысл: скорость роста производительности. Если производительность команд растет на 5–10% в квартал, а одной из команд — на 30%, можно провести анализ производительности такой команды и выяснить, есть ли у нее какой-то рецепт, которому другие команды могут поучиться. Вам по-прежнему нужно будет учитывать изменения состава команды или прочие факторы, не относящиеся к производительности, которые могли повлиять на изменения.

Выводы об улучшении производительности

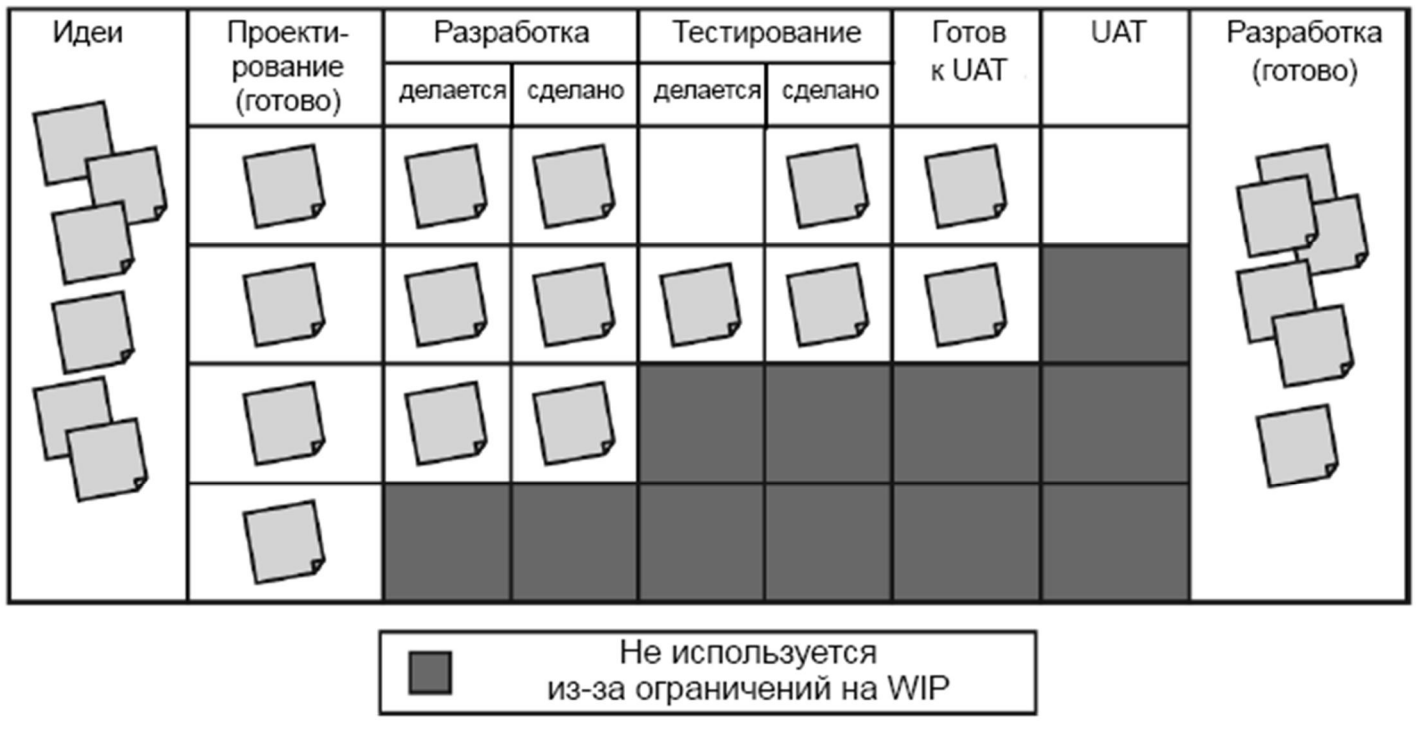
Измерение производительности в отрасли ПО — это минное поле, но несовершенство этих измерений не говорит о том, что их нельзя использовать. При вдумчивом применении измерения производительности позволяют добиться ее резкого роста.

Дисциплина при создании карт историй и контроль выполняемой работы

По мере того как организация выходит за пределы основ Скрама, все бˆольшую пользу приносит применение Lean (бережливого подхода) как способа улучшения качества и производительности. Kanban — это метод, который чаще всего используется в Lean, когда нужно наглядно представить движение рабочего процесса через поток создания ценности.

Kanban делает упор на оценку «незавершенной работы» (WIP, work in progress), определение количества WIP в системе на текущий момент и на постепенное наложение ограничений на WIP, чтобы выявить задержки, которые ограничивают пропускную способность.

При работе по Kanban часто используют специальную доску:



В переводе с японского Kanban означает «вывеска», или «рекламный щит». Задачи пишутся на стикерах и клеятся на доску. Задачи передвигают слева направо, но работу вытягивают с правой стороны, а не выталкивают с левой, насколько это позволяет открытое пространство на доске. На доске, показанной на рисунке выше, задачу можно вытянуть на этап пользовательского приемочного тестирования или внутреннего тестирования, но на ней не хватает пространства для того, чтобы вытянуть работу в какое-нибудь другое состояние.

С точки зрения концепции бережливого производства работа всегда попадает в одну из трех категорий:

- *Ценность* — работа, которая сразу по выполнении приносит определенную ценность, то, за что клиент готов платить деньги.
- *Необходимые издержки* — работа, которая не приносит ценности сама по себе, но представляет собой вспомогательную работу, необходимую для получения такой ценности, например тестирование, закупка лицензий и так далее.
- *Излишние издержки* — работа, которая не приносит ценности, ухудшает пропускную способность и которую можно сократить.
- Функция ограничений на WIP заключается в выявлении времени простоя, которое является крупным источником издержек при выполнении проектов по разработке. Приведем примеры случаев времени простоя:
- Ожидание завершения ручного приемочного тестирования, после того как код прошел юнит-тестирование, интеграционное тестирование и регистрацию изменений, прежде чем можно будет развернуть функционал.
- Ожидание обнаружения ошибок независимыми тестировщиками для их последующего исправления командой разработчиков, прежде чем можно будет развернуть ПО.
- Ожидание завершения код-ревью, прежде чем получится завершить историю в спринте.
- Ожидание, пока команда на одной площадке зарегистрирует изменения в коде, которые необходимо внести, прежде чем команда на другой площадке сможет продолжить свою работу.
- Ожидание, пока владелец продукта уточнит историю, прежде чем команда разработчиков сможет начать реализацию истории.

- Ожидание решения исполнительного руководителя, которое определяет, каким путем пойдет команда.

Так или иначе, время простоя в проектах по разработке откладывает сроки реализации функциональности и, следовательно, всегда представляет собой издержки.

Когда команды в первый раз создают карты рабочих потоков, то обнаруживают, что у них слишком много WIP.

Фокус на количестве WIP выявит, что пропускная способность редко связана с максимальной нагрузкой отдельных людей. Сильное желание загрузить каждого сотрудника до полной занятости чаще создает узкие места, снижающие производительность. Внимание к количеству WIP может быть невероятно полезно в том, чтобы помочь организации перейти от максимального увеличения занятости до максимального увеличения пропускной способности.

Подробное обсуждение Kanban и Lean выходит за рамки этой книги. Дополнительная литература по теме приводится в конце этой главы.

Ретроспективы в Agile

Ретроспектива — это встреча, представляющая собой главный отрезок времени, на котором рассматриваются новые улучшения и оцениваются уже проведенные улучшения. В скрам-проектах ретроспектива спринта проводится в конце спринта после его обзора и перед планированием следующего.

Цель ретроспективы — изучить, как прошел спринт, выработать идеи по улучшению производительности, оценить уже реализованные идеи по улучшению производительности, выработанные на предыдущих ретроспективах, и создать план по реализации улучшений производительности во время следующего спринта. Скрам-мастер проводит фасилитацию встречи, участвует вся команда.

В общих чертах встреча проходит так:

1. Подготовка. Предложите идеи по улучшению мышления, ориентированного на рост, напомните всем о том, что нужно сосредоточиться на работе над системой. Есть компания, начинающая каждую ретроспективу с шутки, которая задает тон декриминализации ошибок и психологической безопасности.
2. Сбор входных данных; создание общего пула информации.
3. Получение выводов, поиск закономерностей и основных причин, обзор общей картины.
4. Принятие решений, определение экспериментов, которые нужно провести команде, создание плана действий.
5. Закрытие ретроспективы, включая обзор того, как можно улучшить саму ретроспективу.

Повышенное внимание к ретроспективам может касаться любых областей, работа над которыми может позволить улучшить производительность в течение следующего спринта, в том числе:

- Процессы и методы.
- Взаимодействие.
- Среда.
- Продукты.
- Инструменты.

Время, выделяемое на ретроспективу, ограничено. Как правило, длина ретроспективы для двухнедельного спринта составляет 75 минут.

В разных командах по-разному относятся к присутствию сторонних участников или наблюдателей на ретроспективах. Менеджеры всегда могут пересмотреть планы по улучшению производительности, которые следуют из ретроспективы, но я считаю, что полезнее добиться на самой ретроспективе наибольшей откровенности, чем приглашать сторонних наблюдателей.

Дайте время изменениям вступить в силу

В настоящее время в Скраме практикуется подход, при котором каждая ретроспектива приводит минимум к одному изменению в следующем спринте. Последствия такого изменения рассматриваются на последующей ретроспективе, в результате изменение сохраняют или отменяют. Изменения могут поступать в бэклог продукта и входить в план на будущий спринт.

Желание не дать команде успокоиться — разумно, но я думаю, что в этом метод заходит слишком далеко. Важно находиться в равновесии с необходимостью измерять последствия каждого изменения. Слишком много изменений за очень короткий промежуток времени не дадут понять, как изменения оказали влияние на скорость.

Выделяйте некоторое время на стабилизацию среды после внесения изменений, чтобы понять действие каждого изменения. Сразу после применения изменения могут вызвать падение производительности перед ее ростом, это нужно учитывать.

Обзор начисления единиц сложности историй на ретроспективах

Количество единиц сложности историй не изменяется после их присвоения, но можно провести их обзор на ретроспективах. Если команда соглашается в том, что размер истории отличается на одно число Фибоначчи по сравнению с присвоенным (то есть было присвоено значение 3, но оказалось, что размер 5), присвоенное значение считается достаточно точным. Если присвоенное количество отличается больше, чем

на одно число, то это ошибочная оценка. Отследите, сколько историй оценено ошибочно.

Количество ошибочных значений можно использовать как показатель того, достаточно ли команда уточняет бэклог перед начислением единиц сложности, в достаточной ли мере она проводит декомпозицию историй, достаточно ли подробно она обсуждает истории во время планирования спринта и так далее.

Не превращайте измерения в игру

По мере того как вы работаете над улучшением процесса, проверяйте, чтобы все улучшения были подлинны, не просто были изменениями способов измерения или состава команды.

Разные команды используют разные подходы к тому, как начислять единицы сложности историй (это одна из причин, почему сравнение разных команд настолько сложное, а сравнение разных компаний вовсе бессмысленно). Некоторые команды используют единицы сложности для работы над устранением дефектов, а некоторые нет. Некоторые измеряют единицами сложности историй спайки, некоторые так не делают. По моему опыту, какие-то варианты работают лучше других, но что-то не работает в принципе — изменение характера работ, измеренных единицами сложности историй, вместо подлинного улучшения процесса.

Если оказывается, что команду одолевает азарт при измерениях, рассмотрите это как возможность декриминализировать ошибку. Рассмотрите поведение с точки зрения систем и исправьте ту систему, которая вызывает проблему. Согласно факторам профессионализма в теории автономии, профессионализма и цели, команды, как правило, хотят улучшить результаты. Если вы обнаружите команду, которая азартно играет с системой, а не применяет измерения для улучшения производительности, выясните, что подрывает естественное желание команды совершенствоваться. Может, давление сроков? Нехватка времени для размышления и адаптации? Нехватка разрешений на внесение в процесс изменений, которые приведут к улучшению показателей? Это одна из возможностей подумать о вашей работе как руководителя и оценить влияние своей работы на команды.

Изучайте и приспосабливайтесь

Помимо формальной ретроспективы образ мышления «изучайте и приспосабливайтесь» нужно применять в проекте от начала до конца. Скрам предоставляет несколько структурных возможностей для применения принципа «изучайте и приспосабливайтесь»:

- Планирование спринта.
- Обзор спринта.

- Ретроспектива спринта.
- Любой случай, когда недочет обнаружен по завершении соответствующего спринта.

Эффективность использования принципа «изучайте и приспосабливайтесь» зависит от некоторой степени *нетерпимости*. Команды, в которых долго терпят проблемы, не развиваются. Команды, которые настойчиво пытаются решать свои проблемы, могут невероятно быстро вырасти.

Эффективность применения принципа «изучайте и приспосабливайтесь» также можно повысить благодаря структуре и прозрачности. Мы успешно работали с командами благодаря тому, что они помещали предложенные изменения в свой бэклог продукта, ставили приоритеты и планировали работу по улучшению процесса наряду с другими работами. Это помогает избежать ошибки, когда результаты ретроспективы появляются только «на бумаге», и позволяет избежать проблемы, когда слишком много изменений производится одновременно.

Прочие соображения

Измерение производительности отдельных специалистов

Производительность отдельных людей пытались измерить во многих отраслях, в том числе в медицине, образовании и разработке. Нет универсального верного способа измерить производительность отдельных специалистов. Лучшие врачи могут вести наиболее тяжелых пациентов; у них может быть более низкий процент вылеченных, чем у других врачей, даже несмотря на то, что они лучшие. Лучшие учителя могут работать в самых трудных школах, и их ученики могут получать более низкие оценки по тестам, несмотря на то что их учителя лучше. Лучшие разработчики могут браться за наиболее сложную работу, и в таком случае их видимая производительность будет ниже, чем у средних разработчиков.

Различия есть в сложности заданий, многозадачности при ведении нескольких проектов, межличностной динамике взаимодействия с остальными членами команды, уровне поддержки заинтересованными сторонами своих проектов, времени, потраченном на обучение других членов коллектива, а также в других бесчисленных факторах, которые влияют на отдачу от каждого отдельного специалиста. Вне исследовательских условий в проектах по разработке содержится слишком много переменных, мешающих получению значимых измерений производительности отдельных людей.

Agile большее значение придает командам, а не отдельным специалистам. Измерения производительности на уровне команд больше соответствуют культуре Agile и гораздо точнее.

Рекомендации

Изучайте

- Достаточно ли последовательны практические методы вашей команды при работе по Скраму, чтобы сформировать эталон, по которому можно проводить измерения?
- Проведите обзор производительности команд на обзорах спринта, ретроспективах и при планировании спринтов. Используют ли они преимущества этих возможностей, следуют ли принципу «изучайте и адаптируйтесь»?
- Насколько хорошо вы как руководитель поддерживаете команды в улучшении процесса, особенно когда ищете равновесие между краткосрочными потребностями доставки и долгосрочными задачами по улучшению процесса?
- Создайте карту рабочих потоков и поищите задержки. Оцените, сколько издержек возникает у вас в процессе доставки из-за ненужных задержек.

Приспосабливайтесь

- Начните измерять последствия изменений процесса с помощью единиц сложности историй.
- Поощряйте последовательное использование вашими командами принципа «изучайте и приспосабливайтесь» во время соответствующих скрам-мероприятий.
- Предусмотрительно сообщите вашим командам, что ретроспективы важны и что вы поддерживаете свои команды в немедленном внесении изменений в следующем спринте исходя из результатов проведенных ретроспектив.
- Представьте наглядно работу ваших команд с помощью Kanban и поищите задержки.

Дополнительные ресурсы

[Эстер Дерби и Диана Ларсен, 2006]. Agile Retrospectives: Making Good Teams Great. Это материал о том, как проводить ретроспективы Agile

[Маркус Хаммарберг и Йоаким Санден, 2014]. Kanban in Action. Хорошее введение в Kanban в контексте разработки ПО.

[Том и Мэри Поппендик, 2006]. Implementing Lean Software Development. Это еще одно введение в Kanban и бережливое производство с упором на программное обеспечение.

[Дантар П. Остерваль, 2010]. The Lean Machine: How HarleyDavidson Drove Top-Line Growth and Profitability with Revolutionary Lean Product Development. В этой книге описано тематическое исследование о том, как *Harley-Davidson* улучшила процесс разработки продукции посредством применения концепции бережливого производства.

[Стив Макконнелл, 2011]. What does 10x mean? Measuring Variations in Programmer Productivity. Эта глава в книге Making Software: What Really Works, and Why We Believe It описывает различия в производительности отдельных разработчиков и предоставляет более подробное описание трудностей, возникающих при проведении измерений в коммерческой среде.

[Стив Макконнелл, 2016]. Measuring Software Development Productivity (сетевой вебинар). Вебинар приводит больше подробностей об измерении производительности команды.

ГЛАВА 20. Еще более эффективная предсказуемость

Несколько десятилетий назад Том Глиб задал вопрос: «Вы хотите предсказуемости или контроля?» [Глиб, 1988]. Без лишней помпы Agile в организациях изменил ответ на этот вопрос. При последовательной разработке, как правило, определяли фиксированный набор функций и затем оценивали расписание — основное внимание уделялось прогнозированию графика работ. При разработке по Agile обычно определяют фиксированный график работ, а затем — наиболее ценный функционал, который можно доставить в необходимые сроки. При этом основное внимание уделяется контролю набора функций.

Множество работ по Agile сосредоточено на разработке ПО для рынков, в которой больший приоритет отдается своевременности, а не предсказуемости — потребительских мобильных приложений, игр, облачных сервисов (*Spotify*, *Netflix*, *Etsy*и так далее). Но что делать, если ваши клиенты все еще хотят предсказуемости? Что делать, если компании нужно доставить определенный набор функций и ей все-таки нужно знать, сколько времени займет доставка такого набора функций? Или что, если вы просто хотите получить представление о том, сколько функционала можно доставить за примерное количество времени, что поможет оптимизировать функционал и график?

В Agile чаще всего делается упор на контроле набора функций, но с помощью правильно выбранных методов можно добиться прекрасной предсказуемости.

Предсказуемость на различных этапах цикла релиза

Практические методы, характерные для Agile, недоступны в самом начале проекта. Перед наполнением бэклога продукта практические методы, используемые для оценки проекта в самом начале, будут теми же независимо от того, будет ли проект выполняться по Agile или модели последовательной разработки [Макконнелл, 2006]. Значимость отличий между Agile и последовательной разработкой становится важной не ранее, чем команда начинает выполнение работ в спринтах.

На рис. 20.1 показана точка, с которой в проекте приобретают значимость методы оценки, характерные для Agile, выраженные в терминах конуса неопределенности.

Существует исключение для этого паттерна, которое заключается в том, что если вы хотите получить сочетание предсказуемости и контроля, а не

чистую предсказуемость, то Agile-методы нужно задействовать немного раньше.

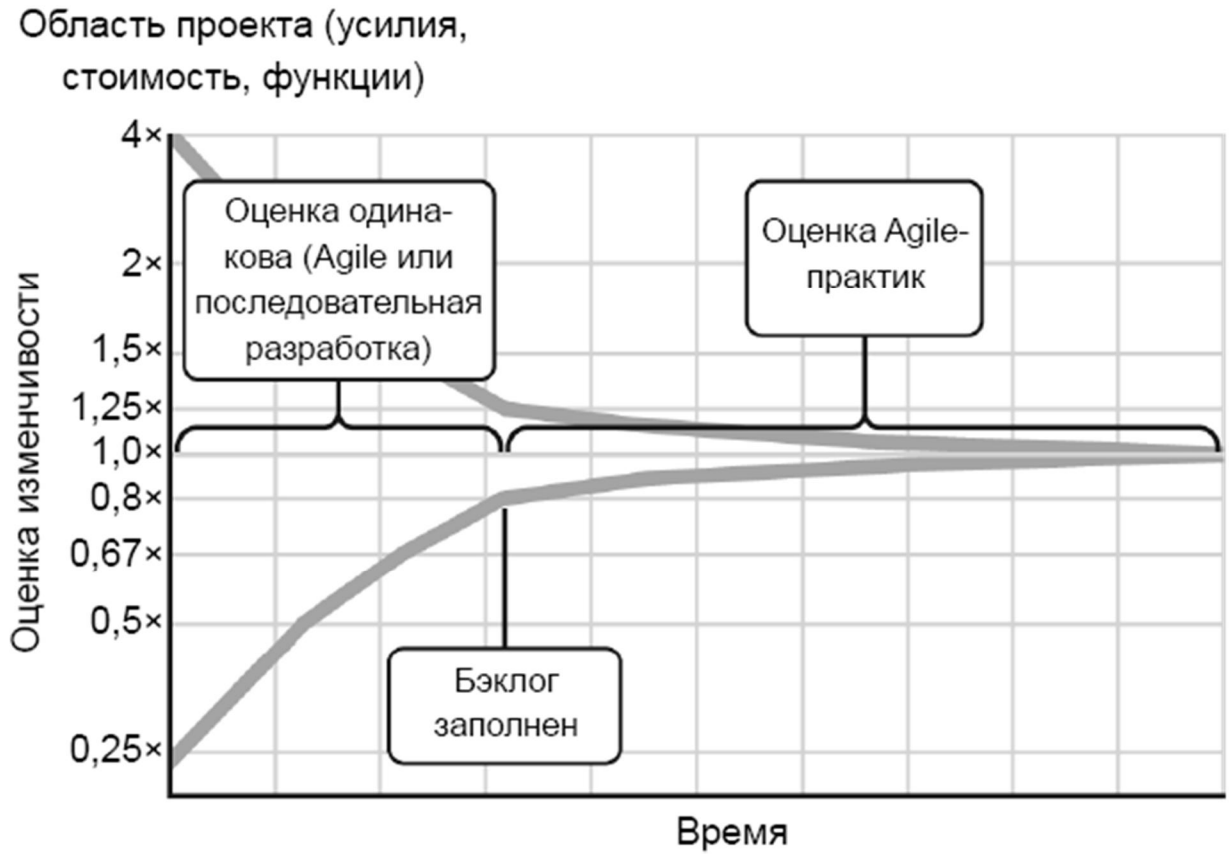


Рис. 20.1. Практические методы оценки по конусу неопределенности. Методы, характерные для Agile, задействованы после заполнения бэклога. Адаптация исследований [Макконнелл, 2006]

Виды предсказуемости

Подходы к обеспечению предсказуемости представлены и кратко описаны ниже. Более подробно о них рассказано в последующих разделах.

Строгая предсказуемость затрат и графика работ. Иногда нужно предсказать затраты и график работ для реализации точного набора функций. Возможно, вы воссоздаете точный набор функциональности на новой платформе. Возможно, разрабатываете определенный набор функциональности для устройства, которое уже выпущено. Или разрабатываете софт не по Agile. Все из перечисленных сценариев касаются предсказуемости и преуменьшения важности контроля набора функционала. Это не самые распространенные случаи, но время от времени они происходят.

Строгая предсказуемость функций.Иногда нужно точно предсказать, какие функции будут готовы в фиксированные сроки при фиксированном бюджете. Это разновидность первого сценария, и методы будут схожими.

Нестрогая предсказуемость.Иногда требуется предсказать осуществимость сочетания примерных функционала, затрат и графика работ. Ни один из этих параметров не фиксирован строго, в каждом из них допускается небольшая гибкость. Такой вид предсказуемости часто требуется во время составления бюджета, когда вы пытаетесь оценить, существует ли бизнес-обоснование для разработки слабо определенного функционала. Нестрогая предсказуемость также часто требуется для отслеживания прогресса, когда проект уже выполняется. Нестрогой

предсказуемости можно достигнуть через итеративный процесс, который включает в себя прогнозирование и контроль.

В следующих двух разделах описано, что требуется для достижения строгой предсказуемости. Даже если вам не нужна строгая предсказуемость, соображения, которые касаются ее достижения, важны для достижения нестрогой предсказуемости, о которой рассказано в последнем разделе.

Строгая предсказуемость затрат и графика работ

Если нужна предсказуемость затрат и графика работы при точном фиксированном наборе функций, то она имеет смысл после того, как точный набор функций был определен — обычно когда цикл релиза пройден на 10–30%.

Следующие практические Agile-методы способствуют строгой предсказуемости:

- Начисление единиц сложности историй.
- Вычисление скорости.
- Небольшой размер историй.
- Заблаговременное заполнение, оценка и уточнение бэклога продукта.
- Короткие итерации.
- Сгорание задач до релиза.
- Учет изменения скорости.

Если строгая предсказуемость затрат и графика работ не нужна, вы можете перейти сразу к следующему разделу «Строгая предсказуемость набора функций». На некоторые концепции в этом разделе ссылаются в дальнейших разделах, поэтому советую по крайней мере просмотреть заголовки, прежде чем читать дальше.

Поддержка предсказуемости: присвоение единиц сложности историй

Прямая оценка затрачиваемых усилий подвержена предвзятости и субъективности [Макконнелл, 2006]. Предвзятость — это намеренная подстройка оценки под желаемое направление. Субъективность — это ненамеренная подстройка оценки из-за принятия желаемого за действительное или нехватки навыка. В отрасли разработки оценки почти всегда оптимистичны, что приводит к тенденции команд и отдельных людей недооценивать ситуацию.

Единицы сложности историй отчасти полезны потому, что на них не влияет предвзятость. Вместо того чтобы оценивать затраты усилий напрямую, команды применяют единицы сложности, чтобы оценить относительный размер рабочих задач. Люди часто будут переводить в уме

рабочие часы в единицы сложности, когда будут оценивать работы в единицах сложности историй, но при таком переводе оценка не становится хуже благодаря тому, как они применяются. Единицы историй применяют для измерения скорости, которая рассчитывается эмпирически на основе фактической производительности. Команда может оптимистично подумать: «Мы выполним 100 единиц сложности за этот спринт». В конце спринта выяснилось, что она выполнила 50 единиц сложности, а не 100, следовательно, ее скорость — это 50, а не 100. От этого надо отталкиваться при дальнейшем планировании.

Поддержка предсказуемости: вычисление скорости

Наиболее распространено использование показателей скорости для планирования спринта, один спринт за один раз. В той же мере ценно использование показателей скорости для способствования предсказуемости. Если команда работала устойчивыми темпами и выполняла 50 единиц сложности историй за спринт в течение последних трех спринтов (средняя скорость 50), то она может использовать средний показатель скорости, чтобы предсказать, когда она доставит полный объем функционала.

Предположим, компания планирует релиз, рассчитанный на год и составляющий 1200 единиц сложности историй. График работ на год позволяет провести 26 двухнедельных спринтов. Команда работает 8 недель (4 спринта) и видит, что ее средняя скорость — 50 единиц сложности историй за спринт. После этого можно предсказать, что команде понадобится 1200/50 = 24 спринта для завершения запланированной работы. Команда, скорее всего, может доставить требуемый набор функций в течение одного года.

В этом утверждении есть несколько «если», «и» и «но». Истории, которые используются для определения скорости команды, нужно выполнить на 100% — они должны полностью соответствовать строгим критериям готовности.

Также важно, чтобы команда не накапливала технический долг, который нужно будет погасить позже во время цикла релиза, потому что он будет тормозить скорость в дальнейших спринтах. При расчете скорости нужно учитывать отпуск и нерабочие дни. В планах нужно учитывать работу любого характера, которую все еще нужно выполнить после соответствия критериям готовности, например пользовательское приемочное тестирование, системное тестирование и так далее. При расчете скорости команды также нужно учитывать ее изменчивость от спринта к спринту (об этом позже). Но в сравнении с традиционной для последовательной разработки оценкой проектов способность команды проводить измерения производительности на основе эмпирически полученных данных в начале

цикла релиза и использовать измерения для прогноза срока выполнения — большое преимущество.

Поддержка предсказуемости: небольшой размер историй

Как было сказано в главе 18 «Еще более эффективные измерения», небольшой размер историй помогает проводить измерения прогресса в Agile-проектах.

Поддержка предсказуемости: заблаговременное заполнение, оценка и уточнение бэклога продукта

Команде, которой нужна строгая предсказуемость, нужно заранее заполнять бэклог продукта полным набором историй для релиза, то есть применить для заполнения бэклога подход, более характерный для последовательной разработки.

Ей не нужно уточнять истории в таких подробностях, как бы пришлось это делать при чистой последовательной разработке. Уточнять истории достаточно, для того чтобы начислять единицы сложности для каждой задачи бэклога, что больше, чем при типичном подходе Agile. Затем каждой задаче бэклога присваиваются единицы сложности историй.

Сложно проработать каждую историю в достаточных подробностях, для того чтобы в начале проекта имело смысл присваивать единицы сложности историй по шкале от 1 до 13. Я изложу свои соображения о том, как справиться с этой проблемой, далее в этой главе.

Поддержка предсказуемости: короткие итерации

Как я уже писал в главе 18, чем короче у вас итерации, тем быстрее вы получите данные о производительности, которые можно использовать для прогноза прогресса команды.

Поддержка предсказуемости: диаграмма сгорания задач до релиза

Контроль выполняемого процесса по сравнению с первоначальным прогнозом команды проводится органично и вписывается в нормальный ход работ. Команда отслеживает с помощью диаграммы сгорания задач количество единиц сложности историй, выполняемых за каждый спринт. Если скорость команды начнет отличаться от первоначальных 50, то она может сообщить об этом стейкхолдерам и внести поправки в план.

Поддержка предсказуемости: учет изменения скорости

Скорость любой команды будет отличаться в различных спринтах. Если команда в среднем выполняет 50 единиц сложности историй в спринт, то

возможно, что в разных спринтах она выполняла 42, 51, 53 и 54 единицы сложности. Это означает, что при использовании показателей скорости команды для долгосрочного прогноза нужно предусмотреть некоторые отклонения или риски.

Результаты команды за четыре спринта показали стандартное отклонение в 5,5 единицы сложности историй от среднего показателя 50. Вы можете рассчитать доверительный интервал на основе количества завершенных спринтов, чтобы оценить риски для средней скорости команды к концу проекта. И вы можете обновлять расчеты, по мере того как команда выполняет больше спринтов и получает больше опыта.

На рис. 20.2 показан пример использования начальной скорости и доверительного интервала для отображения потенциальной низкой и высокой скорости.

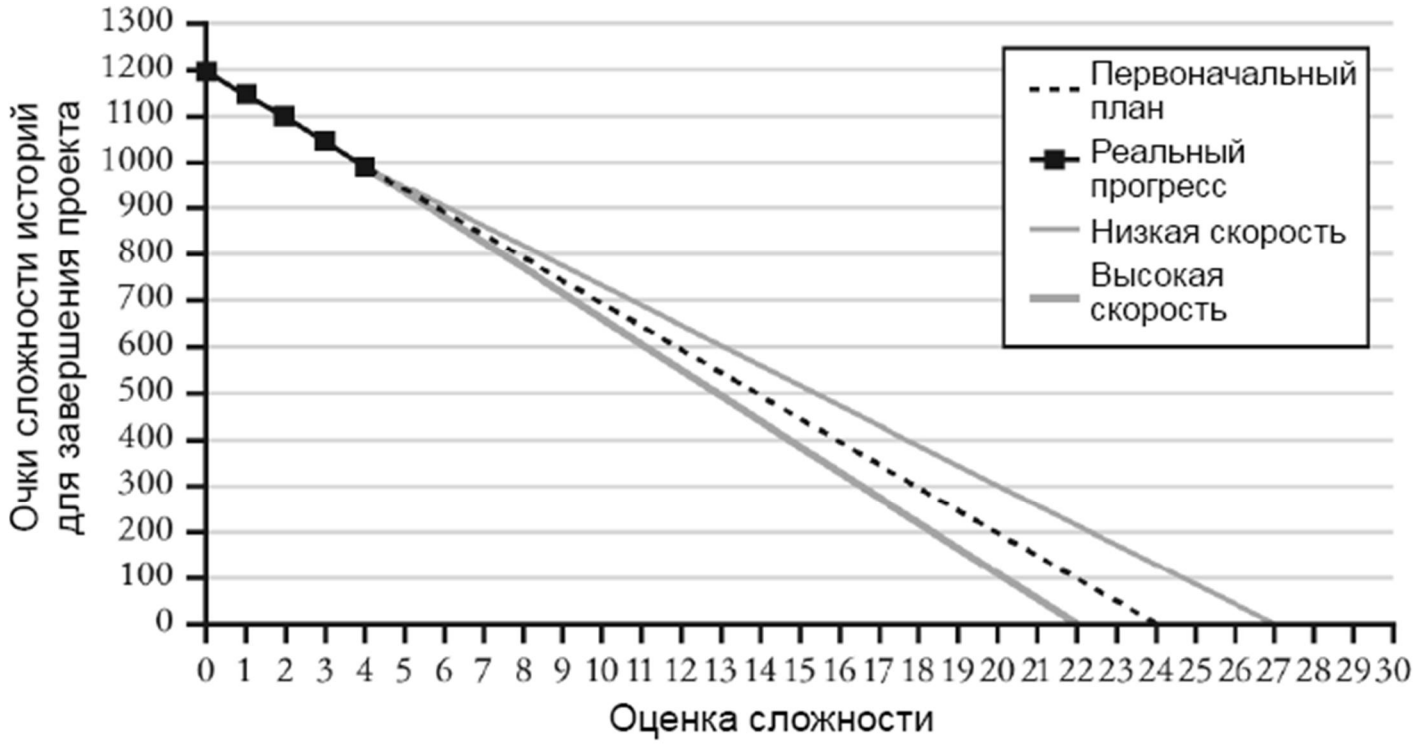


Рис. 20.2. С помощью средней скорости команды, отклонений от скорости и математического вычисления доверительного интервала можно рассчитать различные показатели скорости к концу проекта

Как показано на рисунке, на основании доверительного интеревала 90% [7] команда показывает, что ей потребуется всего 22–27 спринтов для выполнения 1200 единиц сложности историй при номинальных 24 спринтах. Команда показывает небольшие колебания скорости, что приводит к узкому разбросу возможных итоговых значений.

Есть риск недельного опоздания, но наиболее вероятно, что команда справится с проектом в течение года.

Благодаря тому как рассчитываются интервалы, чем больше спринтов команда завершает, тем больше будет сужаться диапазон и тем точнее будет предсказуемость. Если в течение следующих четырех спринтов результаты команды будут так же колебаться, как в первых четырех, доверительный интервал с вероятностью 90% сузится до диапазона в 23–26 спринтов для выполнения всей работы.

Работа со скоростью команды никогда не представляет собой упражнения в чистой предсказуемости, потому что одна из целей — помочь командам достичь стабильной скорости. По мере того как команды совершенствуют свои методы, колебания их показателей должны ослабнуть, а предсказуемость — возрасти.

Строгая предсказуемость набора функций

Если у вас фиксированные затраты и сроки и вам нужно предсказать, какие именно функции можно доставить при таких сроках и затратах, подход будет схож с описанным выше. Ниже рассказано, как подход строгой предсказуемости набора функций распространяется на все ключевые практические Agile-методы.

Создание бэклога продукта

Бэклог требуется заполнить целиком так же, как это делается при подходе, используемом для достижения строгой предсказуемости затрат и графика работ.

Если команда определяет и уточняет истории, размер которых в единицах сложности больше того, на который команда может выделить время, какая-то работа по определению и уточнению будет проделана впустую. Чем качественнее команда заполнит бэклог в порядке приоритета, тем меньше возникнет издержек.

Вычисление скорости в целях прогноза функциональности

Показатели скорости используются примерно так же, как они применяются для достижения строгой предсказуемости затрат и графика работ. Однако вместо того, чтобы использовать скорость для прогноза даты конца работ, ее можно использовать, чтобы предсказать объем функциональности, который можно доставить (то есть количество единиц сложности историй). Теперь нужно учитывать отклонения показателей набора функций, а не графика работ.

Используя тот же пример, что приведен выше — с графиком работ на один год, — можно применить доверительный интервал, чтобы предсказать количество единиц сложности историй, которые можно выполнить за фиксированное количество спринтов, вместо того чтобы предсказывать количество спринтов, которое требуется для доставки фиксированного количества единиц сложности историй.

На основе доверительного интервала с вероятностью 90% после первых 4 спринтов можно рассчитать, что у команды получится доставить в общем 1158–1442 единицы сложности по завершении 26 спринтов, и шансы того, что команда сможет достичь своей цели в 1200 единиц сложности, велики.

Менее строгие подходы к предсказуемости

До этого описание основывалось на подходе, позволяющем добиться чистой предсказуемости. В какой-то момент во время проекта компании может понадобиться предсказать точное сочетание расходов, графика работ и функционала, который в конечном счете будет доставлен, — без внесения в них крупных изменений. Потребность в этом уровне предсказуемости распространена в некоторых отраслях и лишь иногда появляется в других.

Более распространенная потребность, которая мне встречалась, заключается в обеспечении менее строгого уровня предсказуемости, который позволяет руководству контролировать расходы, график работ и функционал как по отдельности, так и вместе. Как я уже писал, во многих случаях роль оценки не в том, чтобы получить точный до мелочей прогноз, а в том, чтобы получить общее представление о том, можно ли выполнить некий вид работ в определенные временнˆые рамки [Макконнелл, 2000]. Это не совсем прогноз, ведь то, что вы пытаетесь предсказать, постоянно изменяется. Это сочетание прогноза и контроля. Независимо от того, как характеризовать такой прогноз, он может удовлетворить потребность организации в предсказуемости и стать эффективным способом ведения проекта по разработке ПО. Практические методы хорошо поддерживают такую нестрогую предсказуемость.

Нестрогая предсказуемость во время планирования бюджета на верхнем уровне

Некоторые Agile-коучи советуют использовать для планирования бюджета на верхнем уровне более крупные значения единиц сложности историй, например 21, 40, 100, или более крупные числа Фибоначчи например, 21, 34, 55 и 89, даже если их не применяют для оценки подробных историй. По вышеописанным причинам применение этих чисел таким способом неверно с точки зрения строгой предсказуемости. С прагматической, менее строгой точки зрения, использование этих чисел таким способом может принести пользу. Компании просто нужно определиться с тем, что означают такие большие числа.

Большие числа как вспомогательное средство против рисков

Вы можете присваивать числовые значения эпикам (или другим крупным задачам, таким как темы, функции и так далее), осознавая, что более крупные числа привносят немного больше риска в предсказуемость. Рассмотрите соотношение единиц сложности подробных историй и эпиков. Если 5% ваших единиц сложности приходится на эпики, то риск плохой предсказуемости не так велик. Но если на эпики приходится 50% единиц

сложности, то риск возрастает. В зависимости от того, насколько вам важна предсказуемость, это будет либо важно, либо нет.

Используйте эпики как бюджет, когда нужна предсказуемость

Другой подход к оценке эпиков и других крупных задач — использовать численные оценки и рассматривать эти числа как бюджет для подробных работ в каждой из областей. Например, если вы используете шкалу Фибоначчи и команда оценивает эпик в 55 единиц сложности историй, то впоследствии можно считать эти 55 единиц сложности бюджетом, который позволительно выделить на этот эпик.

При подходе, в котором эпик представляет собой своего рода бюджет, когда ваша команда раскладывает эпик на подробные истории, для историй, получившихся из этого эпика, нельзя превышать бюджет в 55 единиц сложности. Команде понадобится поставить приоритеты более подробным историям и выбрать из них те, которые приносят наибольшую ценность для бизнеса в пределах 55 единиц сложности.

Такой подход распространен в других видах работ. Если вы делаете ремонт на кухне, у вас будет общий бюджет, а также подробный бюджет на шкафчики, бытовую технику, мебель, кухонные принадлежности и так далее. Подход с подробным бюджетом работает одинаково хорошо для команд разработчиков. Он дает организации чувство предсказуемости, которое достигается посредством сочетания предсказуемости и контроля.

Время от времени команде не будет хватать своего бюджета — в нашем примере она не может доставить основной запланированный функционал в рамках бюджета в 55 единиц сложности. Это приведет к вынужденным разговорам с представителями бизнеса о приоритете работ и о том, стоит ли увеличить бюджет. Такой вид диалога можно назвать здоровым, а начисление единиц сложности историй облегчает его. Этот способ может не обеспечивать тот же уровень предсказуемости, что подходы для получения строгой предсказуемости, но он может оказаться допустимым или даже предпочтительным, если вы цените постепенное внесение поправок по ходу работ больше, чем чистую предсказуемость.

Прогноз сроков доставки основного набора функций в сочетании с дополнительными функциями

Каким-то компаниям не нужна предсказуемость 100% набора функций. Им нужна уверенность в том, что они смогут доставить основной набор функций в течение определенного промежутка времени, а затем уже доставлять дополнительный функционал на свое усмотрение.

Если команде, приводимой в нашем примере, нужно доставить основной набор функций из 1000 единиц сложности историй, она сможет

предсказать, что выполнит основной набор функций примерно за 20 спринтов (40 недель).

До конца года у команды остается запас в 6 спринтов или 300 единиц сложности историй. Организация может принять на себя долговременные обязательства перед своими клиентами о доставке основного функционала, но все равно оставляет себе возможность для доставки функционала своевременно.

Предсказуемость и граница Agile

Большинство компаний могут применять более свободные подходы, описанные в этой главе, и достигать своих целей в бизнесе. У некоторых организаций больше потребности в предсказуемости, поэтому им нужны более строгие подходы.

Сторонники чистоты Agile станут жаловаться, что проработка бэклога продукта в той степени, в которой это необходимо для заблаговременного присвоения точного количества единиц сложности, — уже не Agile. Но цель заключается не в том, чтобы просто внедрить и применять Agile (по крайней мере, если вас интересует тема этой главы).

Цель в том, чтобы применять практические Agile-методы и прочие, чтобы достигать целей и разрабатывать стратегии вашего бизнеса, в том числе предсказуемость, если в них нуждается ваш бизнес.

Как показано на рис. 20.3, концепция «границы Agile», описанная в главе 2 «В чем заключается отличие Agile?», здесь будет полезна.

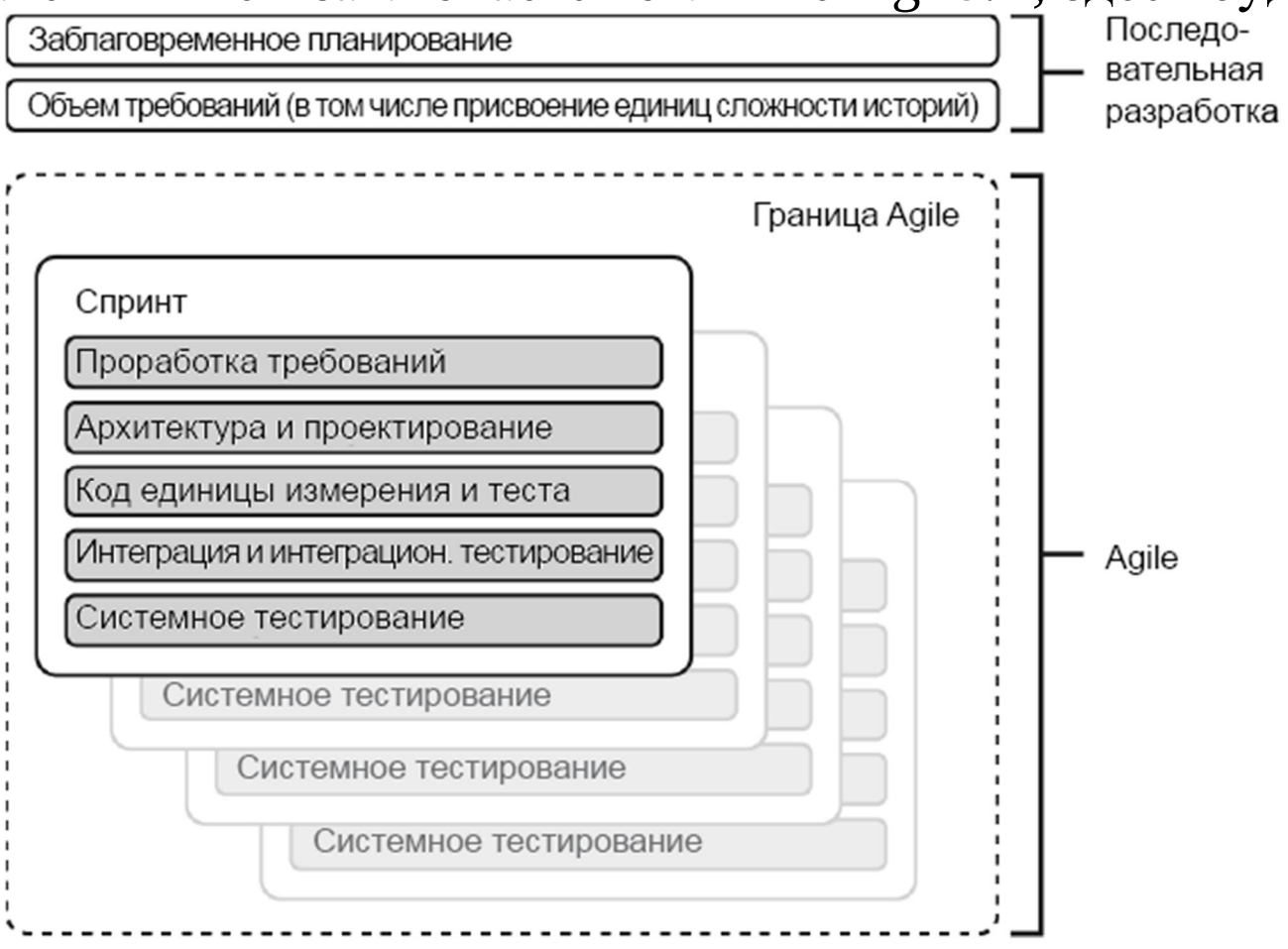


Рис. 20.3. Концепция границы Agile помогает обеспечить долгосрочную производительность организациям, у которых в этом есть необходимость

В целях строгой предсказуемости какие-то действия в начале проекта нужно выполнять с подходами, более характерными для последовательной разработки, а уже потом можно применять Agile в чистом виде.

Предсказуемость и гибкость

Основное внимание в этой главе было сосредоточено на организациях, которым нужна долговременная предсказуемость. Практические Agile-методы прекрасно способствуют достижению этой цели. То, что компании нужна долговременная предсказуемость, не значит, что ей никогда не понадобится менять своих планов. Бизнес, который планирует доставить функциональность на 1200 единиц сложности историй в начале года, иногда может решить изменить курс по прошествии первой половины года. В этом нет ничего страшного. Если команда применяет практические Agile-методы, то сможет оперативно, организованно и эффективно реагировать на изменение курса. Да, в некоторой степени заблаговременная проработка требований окажется ненужной и будет расцениваться как издержки, но было бы еще хуже, если бы команда применяла подход последовательной разработки, полностью прорабатывая каждое требование заранее. Кроме того, команде легче удастся поменять направление благодаря структуре работ, которая представляет собой короткие итерации.

Прочие соображения

Предсказуемость и Cynefin

Полное определение бэклога продукта в начале цикла релиза характерно больше всего для работ в контексте «Сложные» по Cynefin. Если работа находится в основном в контексте «Запутанные», будет невозможно полностью и надежно проработать все задачи до их выполнения.

Напомним, что основное внимание в проекте, который ведется главным образом в контексте «Запутанные», уделяется разведке, необходимой для того, чтобы определить саму суть проблемы, которую нужно решить.

Перед началом полномасштабных работ для исследования проектов, которые имеют выраженные свойства «запутанных», и их превращения в «сложные» предлагались стратегии вроде спиральной модели Барри Боэма [Боэм, 1988]. Этот подход может быть полезен для организации, которая ценит предсказуемость. Но не каждую проблему из «запутанных» можно перевести в «сложные», и ход работ над проблемами, которые по больше мере относятся к «запутанным», никогда не получится хорошо предсказать. Если вам встретился проект, большинство признаков которого указывает на то, что он «запутанный», задайте вопрос, возможна ли хотя бы теоретическая предсказуемость для этого проекта.

Предсказуемость и культура Agile

Предсказуемость может стать щекотливой темой для Agile-команд. Одна из ошибок, которая встречалась нам при внедрении Agile, была в том, что команды отказывались проводить оценки, даже если бизнес излагал здравые причины необходимости в них. Нам встречался не один случай внедрения Agile, который по этой причине накрылся медным тазом.

Нам также встречались примеры того, как сторонники чистоты Agile советовали командам избегать проведения оценок и вместо этого вели коучинг всей организации в том ключе, что в настоящем Agile проведение оценки не нужно. Это примерно то же самое, что хвост начнет вилять собакой, — эти примеры можно рассмотреть как попытки команд разработчиков навязать бизнесу свою бизнес-стратегию.

Одной из первоначальных ценностей, перечисленных в манифесте Agile, является сотрудничество с клиентом. Если вы клиент и вместо того, чтобы предоставлять то, что вы запрашивали, команды разработчиков, работающие по Agile, будут настаивать на том, что вашему бизнесу нужно пересмотреть свои потребности, вы можете предложить пересмотреть отношение к этой ценности Agile.

Рекомендации

Изучайте

- Насколько у вашего бизнеса выражена потребность в гибкости и предсказуемости?
- Вашему бизнесу нужна строгая предсказуемость или достаточно менее строгой?
- Понимают ли ваши команды то, что цель разработки по Agile — обеспечение потребностей бизнеса, и то, что иногда бизнесу требуется предсказуемость?
- Рассмотрите практический метод представления эпиков в качестве бюджета. Как этот подход сработал бы в ваших командах?
- Оцените каждый проект в своем портфолио согласно классификации Cynefin. Приходилось ли командам оценивать работу, которая по своей природе относится к «запутанным»?

Приспосабливайтесь

- Поговорите с командами о том, что бизнесу нужна предсказуемость. Объясните, почему это важно для бизнеса (если важно).
- Оцените каждый «запутанный» проект, можно ли его превратить в «сложный». В проектах, которые остаются «запутанными», переключите основное внимание с прогноза на исследование.
- Попросите команды освежить применение практических Agile-методов, чтобы лучше удовлетворить потребность бизнеса в предсказуемости, в том числе представлять эпики как бюджет.

Дополнительные ресурсы

[Стив Макконнелл, 2006]. Software Estimation: Demystifying the Black Art. В этой книге содержится распространенное рассуждение об оценке разработки ПО в проектах, выполняемых по Agile и по модели последовательной разработки. Она включает в себя многочисленные техники, которые можно использовать при проведении оценки в начале проекта (до того, как проявляется различие Agile и последовательной разработки). С тех пор как в 2006 году эта книга была выпущена, некоторые из описанных в ней взглядов на роль требований в оценке работ были пересмотрены с точки зрения прогрессивного подхода к проработке требований, который описан в этой книге.

ГЛАВА 21. Еще более эффективный Agile в регулируемых отраслях

То, что в начале Agile всеми силами сосредоточился на достижении гибкости, создало впечатление, что практические Agile-методы плохо подходят для применения в регулируемых отраслях, таких как научные, финансовые и правительственные организации. Точка зрения, что «Agile либо полностью, либо никак», укрепила впечатление, что практические Agile-методы не подходят компаниям, которые не могли понять, как привести своих клиентов или общий цикл разработки продукта в полное соответствие с Agile.

Это огорчало, потому что очень много ПО разрабатывается в рамках стандартов, в том числе FDA, МЭК 62304, ASPICE, ИСО 26262, FedRAMP, FMCSA, SOX и GDPR. А другое ПО может казаться нерегулируемым на первый взгляд, но все же подчиняться нормам конфиденциальности, доступа и безопасности.

Когда Agile созрел, оказалось, что практические Agile-методы могут принести такую же пользу во многих регулируемых отраслях, как и во многих других. Можно, конечно, практиковать разработку по Agile так, чтобы она не соответствовала стандартам регулируемых отраслей, но в той же степени можно и так, чтобы соответствовала.

В 2012 году Управление США по надзору за качеством пищевых продуктов и лекарственных средств (FDA) приняло AAMI TIR45:2012 («Guidance on the use of AGILE practices in the development of medical device software» — «Руководство по применению практических Agile-методов при разработке ПО для медицинских приборов») в качестве признанного стандарта. Моя компания более 10 лет работает с многочисленными компаниями, работающими в сфере регулирования FDA и других регулируемых средах, успешно внедряя методы Скрама и Agile.

Материал этой главы относится ко всем регулируемым отраслям, но в особенности к наиболее регулируемым. Нормы FAA/DO-178, в частности, более обширны, чем описано в этой главе, поэтому когда я пишу здесь о «регулируемых средах», я не имею в виду в том числе FAA/DO-178.

Как Agile поддерживает работу в регулируемых средах

В общих чертах, требования, связанные с разработкой ПО, в регулируемых отраслях сводятся к следующему: «Документируйте то, что планируете выполнить; выполняйте то, что планировали; документированно докажите, что работа выполнена». В некоторых средах накладывается дополнительное требование: «Обеспечивайте глубокое отслеживание, чтобы доказать, что работа выполнена на уровне мелочей».

Практические Agile-методы не делают работу над продуктами в регулируемых отраслях сложнее или легче. Документация практических Agile-методов — это большая проблема. Эффективность, с которой можно создавать документацию, — вероятно, самый важный фактор при внедрении Agile-методов в регулируемых средах.

Практические методы последовательной разработки поддерживают создание документации на регулируемые продукты. Акцент Agile на методах, применяемых пошагово и своевременно, приводит к увеличению количества раз, когда документацию нужно создавать или обновлять. Это не всегда проблема. Многие руководители говорили, что разработка по Agile облегчает работу с документацией, потому что с ней можно работать более постепенно, как и с ПО. Тем не менее с некоторых сторон культуру Agile нужно менять, например упор на устную традицию и мудрость племени.

На следующей странице в табл. 21.1 показано, как акценты Agile влияют на соответствие нормативным требованиям в регулируемых средах. На концептуальном уровне некоторые Agile-методы обеспечивают поддержку того, для чего созданы нормы, что гарантирует высокое качество ПО:

- Критерии готовности (которые создаются в соответствии с нормативными требованиями или выходят за их рамки, в том числе в соответствии с требованиями, относящимися к документации).
- Критерии готовности задачи.
- Постоянное обеспечение качества ПО на уровне релиза.
- Разработка тестов либо предшествует разработке кода, либо выполняется сразу после его написания.
- Использование автоматизированных регрессионных тестов.
- Работа по принципу «изучайте и приспосабливайтесь» для улучшения качества процесса и продукта.

Таблица 21.1. Как акценты Agile влияют на работу в регулируемых средах.

Принцип Agile	Применительно к регулируемой среде
Короткие циклы релиза	Сам по себе не влияет на соответствие требованиям, но стоимость каждого релиза может составлять значительную сумму и может влиять на то, насколько часто организация решит выпускать релизы
Сквозная разработка ведется мелкими партиями	Сам по себе не влияет на соответствие требованиям, но влияет на время создания документации
Заблаговременное планирование на общем уровне и своевременное планирование подробностей	Планы нужно обязательно документировать, даже планы, возникающие своевременно, причем может требоваться возможность отслеживания, в зависимости от типа стандарта
Обобщенные предварительные требования вместе со своевременными подробными требованиями	Требования нужно обязательно документировать, даже своевременно возникающие и прорабатываемые; влияет на время создания документации
Стихийное проектирование	Проектирование нужно обязательно документировать, даже своевременное проектирование; влияет на время создания документации
Непрерывное автоматизированное тестирование, интегрированное в разработку	Поддерживает соответствие требованиям
Частое упорядоченное сотрудничество	Некоторые способы взаимодействия нужно перевести с устной традиции на документацию
В общем, подход эмпиричен, ориентирован на оперативность и улучшения	Не влияет на соответствие требованиям

Как Скрам поддерживает работу в регулируемых средах

Нормативные требования могут обновляться редко. Требования в регулируемых средах, которые я описал выше, были изначально созданы десятилетия назад, во времена, когда разработка ПО была чем-то из области фантастики. Организация могла использовать практически любой подход к разработке ПО, и большинство подходов работало не очень хорошо. Нормы предназначены, в частности, чтобы избежать хаотичных, ситуативных практических методов с неизвестным уровнем эффективности.

Федеральные нормативные акты США, как правило, не требуют особого подхода к разработке ПО или особого жизненного цикла. Они требуют, чтобы организации выбрали подход, определили его и задокументировали, как описано выше. Кроме того, иногда они требуют одобрение регулирующего органа.

Agile-методы, особенно Скрам, были формализованы и уже подробно документированы (в том числе в этой книге), что позволяет соответствовать этому требованию. Если команда согласна использовать Скрам и документально подтверждает его использование, что указано в специальном документе, то это вносит вклад в создание определенного процесса, обеспечивающего соответствие нормативным актам.

Использование Скрама в соответствии с требуемыми нормами документации процесса различается, поэтому для наглядности в этом разделе используется стандарт МЭК 62304 («Программное обеспечение для медицинских приборов — процессы жизненного цикла ПО»).

МЭК 62304 требует мероприятий и документации в следующих категориях:

- Планирование разработки ПО.
- Анализ требований.
- Проектирование архитектуры ПО.
- Подробное проектирование ПО.
- Реализация и верификация модулей ПО.
- Интеграция и интеграционное тестирование ПО.
- Системное тестирование ПО.
- Релиз ПО.

Как предполагает стандарт AAMI TIR45, эту деятельность можно сопоставить с моделью жизненного цикла Agile, как показано на рис. 21.1. При помощи этого подхода можно эффективно поделить проекты, выполняемые по Agile в регулируемых средах, на четыре слоя:

- *Слой проекта* — полный набор деятельности для проекта. Проект состоит из одного или более релизов.
- *Слой релиза* — действия, требуемые для создания продукта, готового к использованию. Релиз состоит из одного или более инкрементов (в некоторых регулируемых средах накладываются значительные требования к релизам — например, требование возможности воссоздать точный побитовый образ любого ПО, которое когда-либо было выпущено в течение жизненного цикла устройства, которое приводит к редкому выпуску в релиз).
- *Слой инкремента* — действия, требуемые для создания полезного функционала, но не обязательно продукта, готового к использованию. Инкремент состоит из одной или более историй.
- *Слой историй* — действия, требуемые для создания небольшой части или, возможно, неполного функционала.

При последовательной разработке каждое действие производится чаще всего за один этап. При подходе Agile бˆольшая часть действий распределена по слоям.

При подходе Agile, не характерном для регулируемых отраслей, большинство действий документируется в неофициальном виде. При подходе Agile к регулируемым отраслям действия требуется документировать более формально.

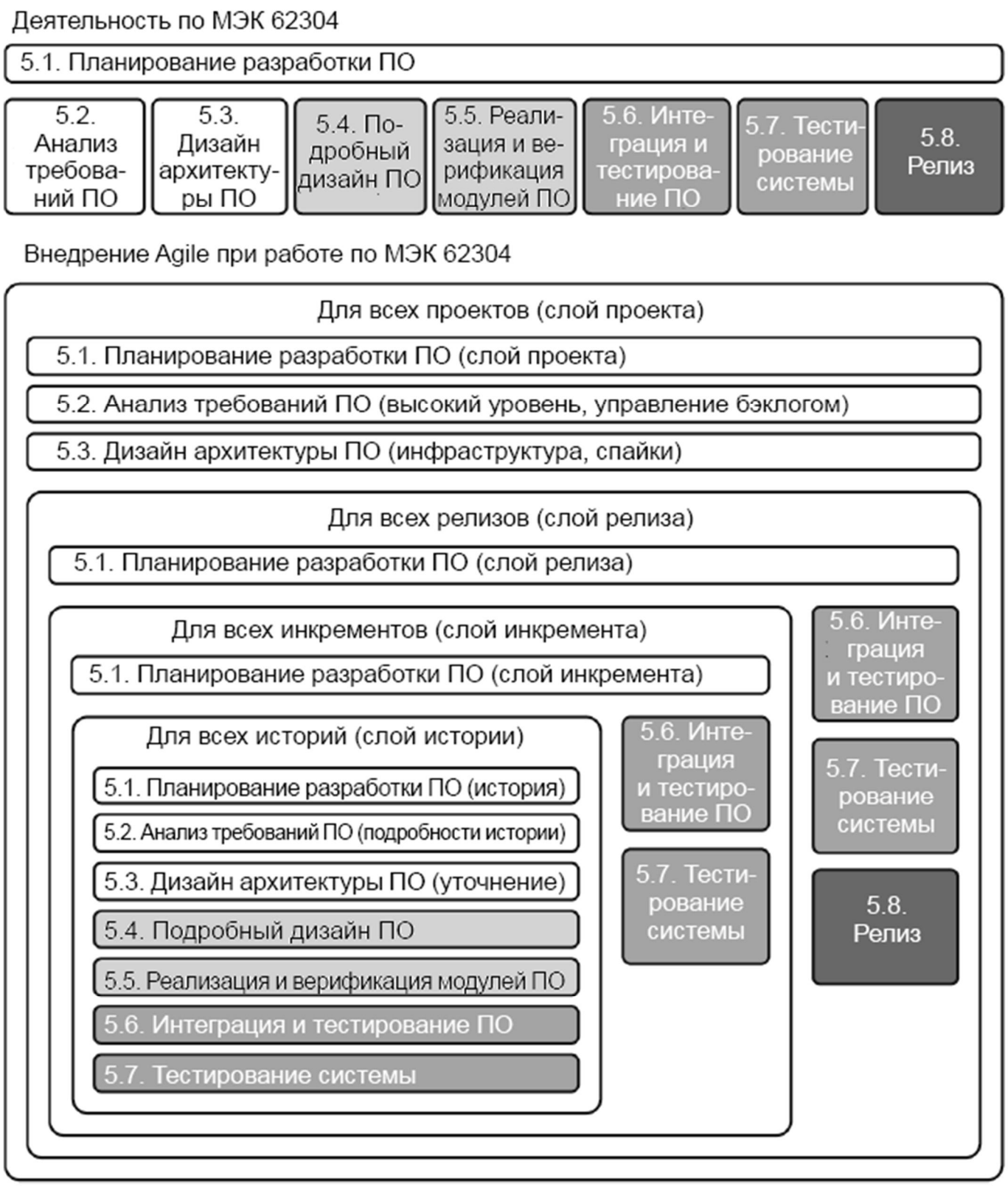


Рис. 21.1. Пример карты категорий нормативной документации процесса при разработке по Скраму. Адаптация исследований [AAMI, 2012]

Отчасти в поддержку эффективного создания документации распределение работы по спринтам потребуется приспособить в соответствии с нормативными требованиями. Отлично показал себя следующий подход:

- Используйте первый спринт (или спринты) для определения общего объема проекта, планирования релиза и заложения основ для архитектуры.
- Проводите нормальные для Скрама спринты по всем правилам. Критерии готовности включают в себя исполнительную документацию для спринта, в том числе создание карты всех пользовательских историй для сопоставления с кодом и тестовыми сценариями.
- При подготовке к релизу проведите спринт по документации, который посвящен доведению документации до готовности, так чтобы она соответствовала нормативным требованиям, в том числе синхронизации требований и проектировочной документации с написанием кода и результатами тестов, а также запуску тестов формальным способом, который позволяет вносить записи о верификации.

Существуют разные варианты этого подхода, о которых я расскажу далее.

Граница Agile для регулируемых систем

Стоимость документации является серьезной проблемой при разработке ПО в регулируемых отраслях, поэтому принести пользу может применение к разработке ПО концепции «границы Agile». Рассмотрите типичный набор действий для разработки ПО.

При отсутствии требований к документации полезным будет применение высокой степени итеративности от планирования до требований и вплоть до приемочного тестирования. Для вас может оказаться полезным оставлять требования на уточнение своевременно, прямо перед началом реализации модуля.

Однако при наличии требований к документации вы можете решить, что обеспечивать высокую степень итеративности при проработке требований слишком затратно, поэтому выгоднее с экономической точки зрения применять подход, более характерный для последовательной разработки. Имея это в виду, вы можете прочертить свою границу Agile после проектирования архитектуры и перед системным тестированием, как показано на рис. 21.2.

В этом случае вы будете использовать преимущественно последовательный подход для планирования, работы с требованиями и проектирования архитектуры, потом перейдете к более инкрементальному подходу к работе по реализации, а затем перейдете обратно к подходу последовательной разработки для системного тестирования ПО.

Рис. 21.2. Пример того, где может проходить граница Agile в проекте, выполняемом в регулируемой отрасли

Сторонники чистоты Agile могут возразить, что этот подход «не настоящий Agile», но опять же — цель не в соответствии Agile на 100%, а в использовании доступных практических методов разработки ПО для обеспечения потребностей бизнеса. Когда вы берете в расчет стоимость производства документации, в регулируемых средах иногда лучше всего срабатывает сочетание подходов Agile и модели последовательной разработки.

Реализация Agile в регулируемой отрасли будет более формальной и структурированной, требовать больше документации, чем при обычной реализации Agile. Команды разработчиков ПО, которые работают в регулируемых отраслях, тем не менее, выиграют от более коротких циклов сквозной разработки модулей, непрерывного тестирования, более тесной обратной связи, частого упорядоченного сотрудничества и сокращения издержек за счет большей доли своевременного планирования и, возможно, своевременной проработки требований и проектирования. Они также могут выиграть от пошагового создания документации.

Прочие соображения

В нашей работе с компаниями в регулируемых отраслях мы обнаружили, что «нормативные требования» не всегда исходят от нормативных актов. Иногда они исходят от неповоротливой политики организации, которая отстает от нормативных требований.

Мы работали с одной научной компанией, которая принуждала к обеспечению отслеживания при проектировании — отслеживания влияния функций до определенных модулей ПО. Мы проанализировали, какие требования к процессу разработки были санкционированы FDA, а какие — регулирующей группой компании. Нам удалось устранить около одной трети проектировочной документации, которая не была санкционирована FDA и фактически была бесполезной.

Мы обнаружили, что требования рассматривались как нормативные требования, которые исходят от опыта компании в проведении аудита клиентов, а не от какого-либо регулирующего органа. Мы также иногда наблюдали, что документация была создана исходя из правил капитализации ПО, а не нормативных требований.

Советую вам обязательно выяснить источники ваших нормативных требований. Поговорите с регулирующими группами, чтобы понять, какие нормативные требования действительно являются таковыми, а какие являются лишь мнением регулирующей группы о том, что нужно клиентам или для финансовых практических методов. Затем можно принять решение

о необходимости перенести исторические требования к документации вашей компании в ваши текущие условия разработки.

Рекомендации

Изучайте

- Узнайте источники нормативных требований в вашей компании. Какие требования действительно вытекают из действующих нормативных актов, а какие — из других источников?
- Пересмотрите способ, по которому в вашей среде создается документация. Можно ли применить практические Agile-методы для сокращения стоимости документации?
- Определите, где сейчас в вашей организации проходит граница Agile, которая разделяет деятельность, связанную с разработкой. Точно ли она проходит там, где нужно?

Приспосабливайтесь

- Если это указано в вашем обзоре документации, создайте план по снижению затрат на документацию, по которому будете создавать документацию постепенно.
- Создайте план переноса границы Agile в вашей организации для лучшего соответствия целям организации, в том числе цели экономической эффективности ведения документации.

Дополнительные ресурсы

[AAMI, 2012]. Guidance on the use of AGILE practices in the development of medical device software. AAMI TIR45 2012. В настоящее время это исчерпывающий справочник для применения Agile в регулируемых отраслях.

[Кит Коллайер и Джорди Мансано, 2013]. Being Agile while still being compliant: A practical approach for medical device manufacturers. Сетевой ресурс (5 марта 2013 года). В этом легком для чтения тематическом исследовании описано, как одной команде удавалось соответствовать нормативным требованиям с помощью подхода Agile.

[Scaled Agile, Inc., 2017]. Achieving Regulatory and Industry Standards Compliance with the Scaled Agile Framework® (SAFe®) Scale Agile, Inc. Исследование, август 2017 года. В этом исследовании описано, как достигать соответствия нормативным требованиям с применением SAFe в качестве особого подхода к Agile. Короткое и содержательное дополнение к этой главе.

ГЛАВА 22. Еще более эффективное управление портфолио

Многие компании ведут портфолио своих проектов в свободной форме. Они используют интуитивные практические методы, чтобы решать, какие проекты первыми начать, а какие — первыми закончить.

Такие компании не понимают, как дорого им стоит неформальный подход к управлению портфолио проектов. Если бы они понимали, то скорее бы предпочли сжигать стопки стодолларовых купюр, чем дальше просиживать штаны, применяя прежние методы управления портфолио проектов.

Разница в пользе между интуитивными подходами к управлению портфолио и математически обоснованными подходами велика, а более короткое время циклов в Agile-проектах создает еще больше возможностей для увеличения ценности, доставляемой благодаря грамотному управлению портфолио.

Более ценная и короткая работа сначала

Основным инструментом управления портфолио в Agile является метод «Более ценная и короткая работа сначала» (Weighted Shortest Job First, WSJF).

Концепция метода «Более ценная и короткая работа сначала» берет начало из работы Дона Рейнертсена по бережливой разработке продукта [Рейнертсен, 2009]. В методологии Agile ее главным образом ассоциируют с SAFe, но концепция широко применима независимо от того, применяет ли организация SAFe.

WSJF начинает с определения «стоимости задержки» (CoD), ассоциированной с каждой функцией или историей. Стоимость задержки — не совсем интуитивное понятие, которое означает издержки, возникающие от упущения возможности из-за отсутствия той или иной функции. Если функция сэкономит вашему бизнесу 50 000 долларов в неделю при выходе в доступ, стоимость задержки составит 50 000 долларов в неделю. Если функция обеспечит вашему бизнесу доход в 200 000 долларов в неделю при выходе в доступ, стоимость задержки составит 200 000 долларов в неделю.

WSJF эвристически минимизирует стоимость задержки для набора функций. Предположим, что у вас есть набор функций (табл. 22.1).

Таблица 22.1. Пример набора функций со сведениями, необходимыми для расчета WSJF

Функция	Стоимость задержки (CoD)	Продолжительность разработки	WSJF: CoD / продолжительность
A	$50 тыс./неделю	4 недели	12,5
B	$75 тыс./неделю	2 недели	37,5
C	$125 тыс./неделю	8 недель	15,6
D	$25 тыс./неделю	1 неделя	25

В соответствии с таблицей начальная общая стоимость задержки составляет 275 000 долларов в неделю — сумма стоимостей задержки всех функций. Когда вы начинаете доставлять функциональность, вы перестаете нести расходы по стоимости задержки функционала, который вы доставили.

Правило WSJF заключается в том, что вы доставляете первым делом функцию с наибольшим WSJF. Если у нескольких задач один и тот же WSJF, сначала выполните ту, на которую уйдет меньше времени.

Предположим, что вы реализовали функции в порядке от наибольшей CoD к наименьшей. Диаграмма общей CoD будет выглядеть, как на рис. 22.1.

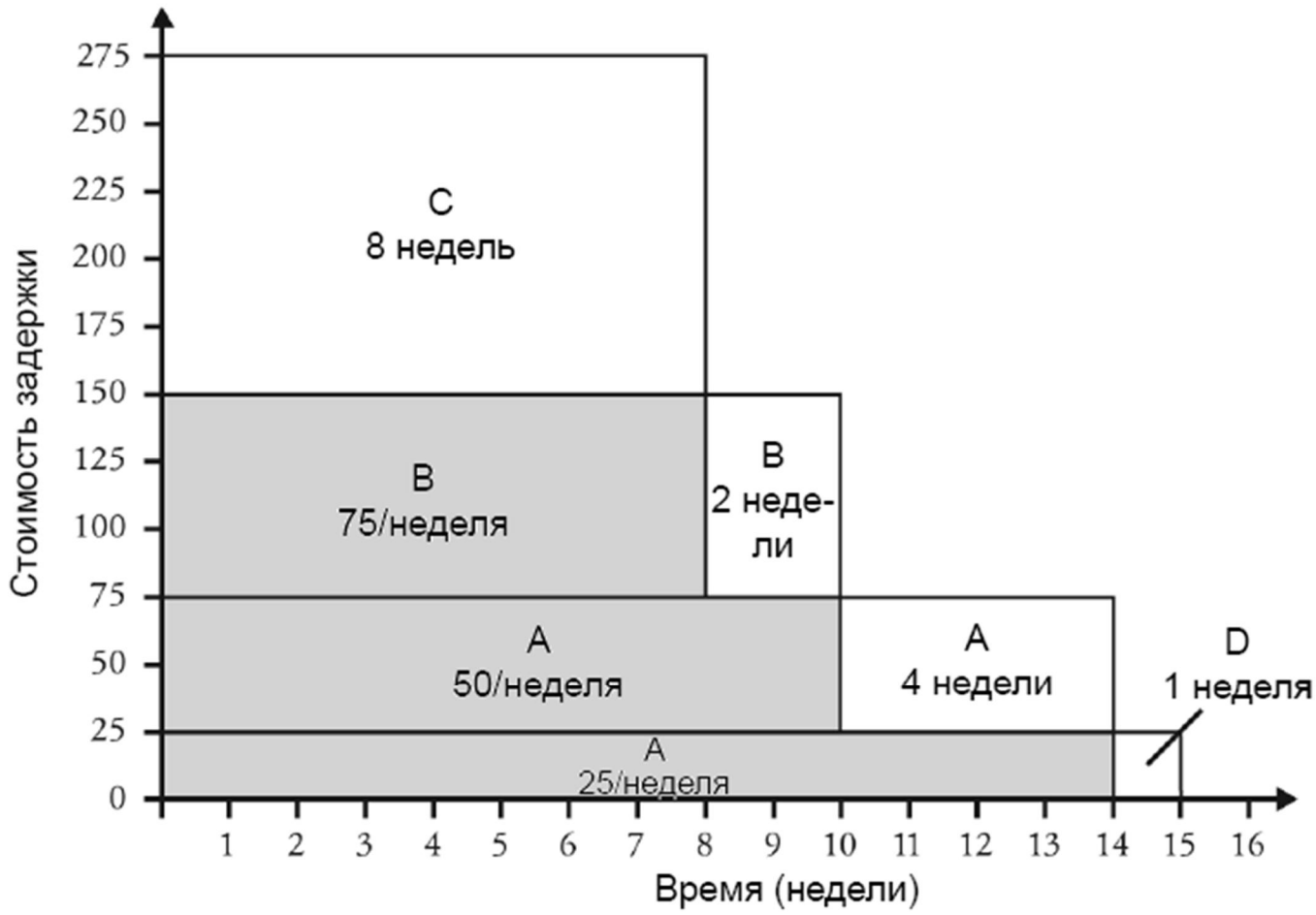


Рис. 22.1. Пример общей CoD для функций, доставленных в порядке убывания CoD

Белые прямоугольники показывают функцию, которая выполняется в текущее время: работа сначала начинается над функцией C (наибольшая CoD), затем над функцией B, A и, наконец, D (наименьшая CoD).

Стоимость задержки, CoD, накапливается для каждой функции до ее выполнения. Общая CoD рассчитывается как общая площадь, занимаемая всеми прямоугольниками, как светлыми, так и темными. В этом примере общая CoD составляет 2,825 млн долларов: 8 недель по 125 000 долларов в неделю для функции C плюс 10 недель по 75 000 долларов в неделю для функции B и так далее.

На рис. 22.2, с другой стороны, показаны функции в порядке убывания WSJF — CoD, разделенной на продолжительность, — а не простой CoD. Пунктирная линия показывает кривую доставки в порядке убывания простой CoD.

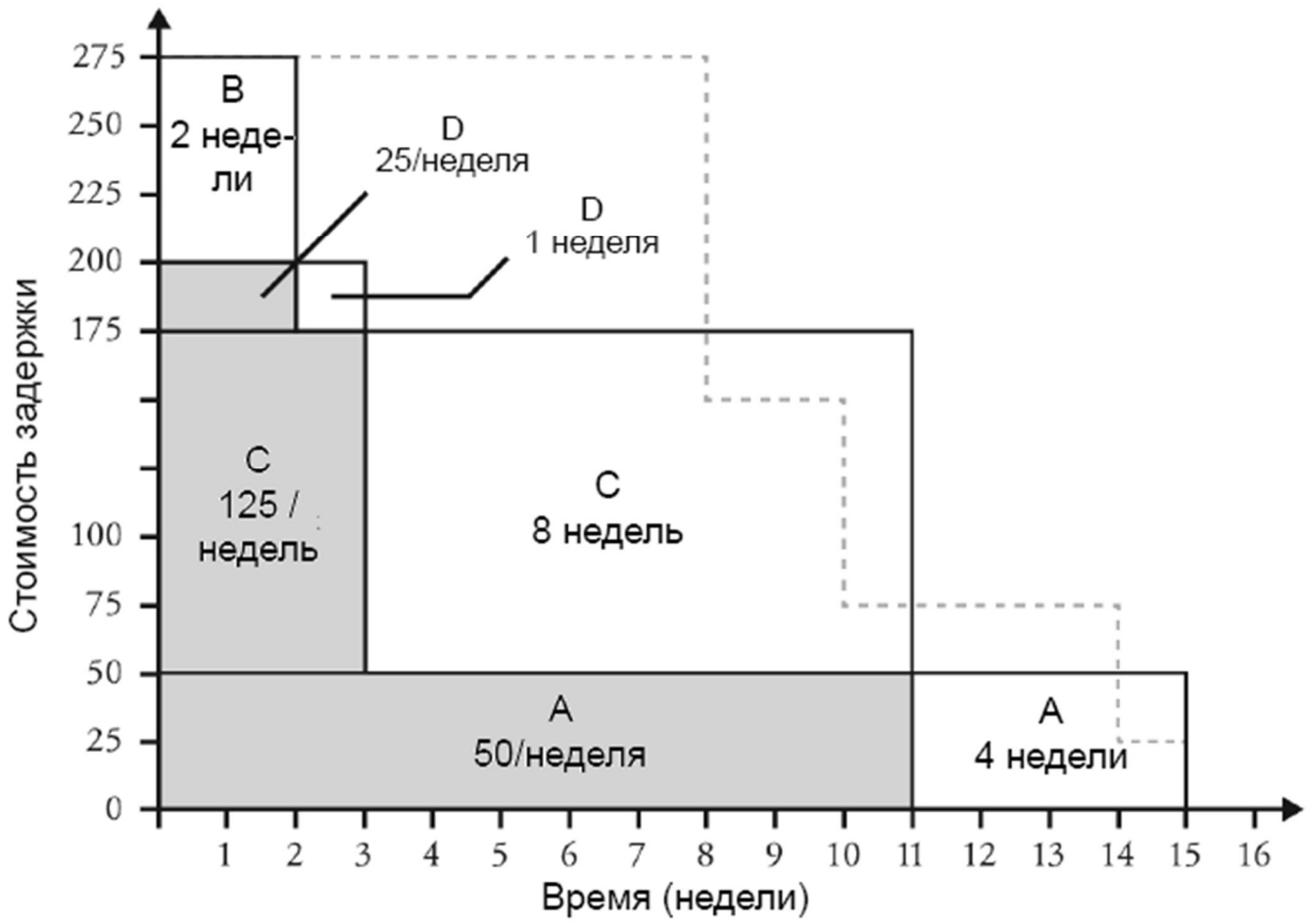


Рис. 22.2. Пример общей CoD для функций, доставленных в порядке убывания WS]F

Вы наглядно можете увидеть, что общая площадь прямоугольников при доставке в этом порядке меньше, чем площадь при доставке в порядке убывания CoD. Математически общая CoD при таком порядке составит 2,35 млн долларов, CoD снизится (а ценность для бизнеса возрастет) на 0,475 млн.

Такого невероятного роста ценности для бизнеса можно достичь, лишь просто изменив порядок, в котором вы доставляете функции!

Распространенная альтернатива, но гораздо более плохая, чем порядок WSJF

Недостаточно оптимальное выполнение функций в порядке убывания CoD распространено, даже несмотря на то, что выполнение в порядке WSJF явно лучше. Существует куда худший порядок доставки, причем распространенный, — одновременная нагрузка на весь цикл выделения бюджета: все четыре функции начинают выполнять в начале цикла и не успевают доставить ни одну из них к концу цикла (рис. 22.3).

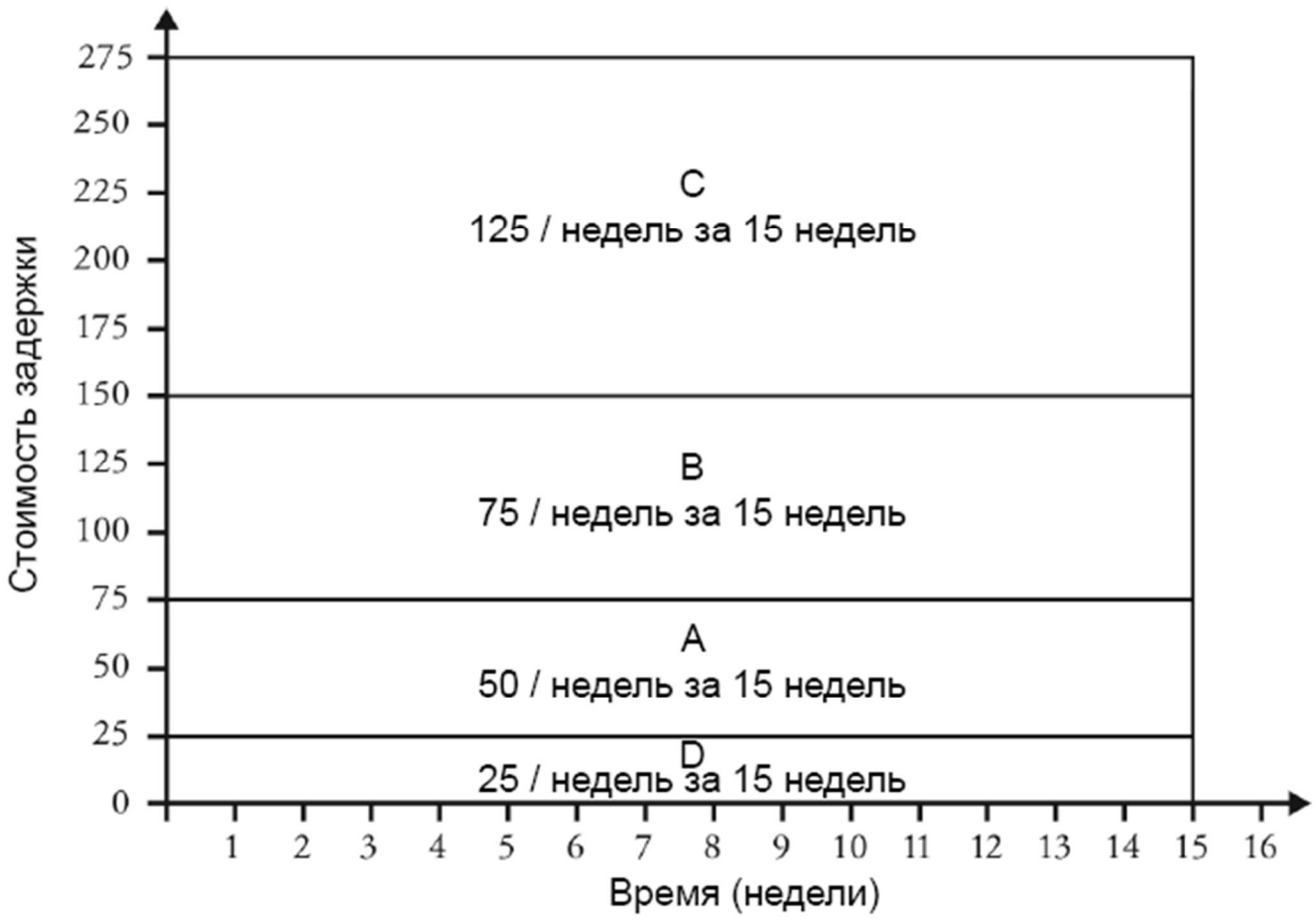


Рис. 22.3. Общая CoD для функций, доставляемых на основе циклов выделения бюджета

Общая CoD при таком подходе составит 4,125 млн долларов, что гораздо хуже, чем при любом из уже упомянутых подходов.

В бережливом производстве есть мантра, которая гласит: «Заканчивайте начинать, начинайте заканчивать». Упущенная возможность, представленная в этом примере, будет не особо заметна в организации, которая работает по модели последовательной разработки по каденциям длиной в квартал или в год. Ее гораздо легче увидеть, когда организация переходит на каденции длиной в неделю или две.

Единицы измерения стоимости задержки, альтернативные деньгам

До этой строчки стоимость задержки в примерах выражалась в долларах. Существуют два обстоятельства, при которых вы можете выбрать другие единицы для вычисления CoD.

Эти единицы — не денежный эквивалент. В средах, где безопасность критически важна, CoD может выражаться в том, что медицинский прибор окажется непригоден для спасения жизней, а экстренные службы не смогут принимать вызовы. В этих условиях можно выражать CoD в жизнях, травмах и других соответствующих единицах. Несмотря на это, WSJF рассчитывается так же.

У вас нет достаточных сведений о стоимости. Более вероятен сценарий, что стоимость исчисляется в деньгах, но нет точных или надежных сведений о стоимости задержки. В этом случае можно использовать относительные единицы измерения. Agile-команды, как правило, используют числа Фибоначчи (1, 2, 3, 5, 8, 13, 21). После присвоения относительной стоимости расчеты WSJF производятся так же.

Прочие соображения

Размеры футболок

Подход с использованием размеров футболок, описанный в главе 14 «Еще более эффективное определение приоритетов требований», можно использовать для планирования на уровне портфолио. Нам доводилось работать с компаниями, которые успешно применяли этот метод. Но если компания способна рассчитывать стоимость задержки для своих работ, особенно если ее можно рассчитать в денежном выражении, значительная ценность для бизнеса, получаемая от использования WSJF, делает WSJF предпочтительным.

Рекомендации

Изучайте

- Подумайте над тем, какой размер функции, требования или проекта будет достаточно большим для поддержки расчета стоимости задержки в вашей организации. Позволит ли применение CoD и WSJF вашим командам улучшить планирование на уровне функций или только планирование на уровне портфолио?
- Изучите портфолио своих текущих проектов, применяя CoD и WSJF. Получите сведения о CoD от ваших бизнес-структур и о продолжительности разработки от команд. Рассчитайте общую CoD ваших текущих приоритетов. Вычислите порядок WSJF для вашего портфолио, а затем посчитайте, какова будет общая CoD, если вы расположите проекты в портфолио в порядке WSJF.

Приспосабливайтесь

- Расположите проекты в портфолио в порядке WSJF.
- Подумайте о применении подхода с WSJF к более мелким задачам, таким как эпики.

Дополнительные ресурсы

[Дональд Г. Рейнертсен, 2009]. The Principles of Product Development Flow: Second Generation Lean Product Development. В этой книге содержится описание CoD и WSJF, а также приведено глубокое рассуждение о теории очередей, размерах партий и увеличении потока.

[Джез Хамбл и др., 2015]. Lean Enterprise: How High Performance Organizations Innovate at Scale. В этой книге также рассказано о WSJF, она больше ориентирована на разработку ПО. В ней WSJF называют CD3 (Cost of Delay Divided by Duration).

[Стив Токи, 2005]. Return on Software: Maximizing the Return on Your Software Investment. В этой книге содержится подробное рассуждение о принятии экономических решений в области инжиниринга, в том числе интересные рассуждения о принятии решений в условиях риска и неопределенности.

ГЛАВА 23. Еще более эффективное внедрение

В других частях этой книги в подробностях описаны практические Agile-методы, которые характерны для внедрения. В этой главе рассказано о самом внедрении, изменении организации.

Независимо от того, боретесь ли вы за Agile, который уже частично внедрили, или только начинаете его внедрять, в этой главе вы найдете материал о том, как внедрить Agile успешно.

Общий подход к изменениям

Интуитивный подход к внедрению Agile кажется незамысловатым:

Этап 1. Начать с пробной команды. Соберите первую команду для пробного внедрения Agile в вашей организации. Проработайте возникающие трудности, работая с одной командой.

Этап 2. Распространить работу по Agile на одну или более команд дополнительно. Разверните применение практических Agile-методов в других командах, применяя опыт, полученный из работы с первой командой. Учредите сообщества практиков для обмена полученным опытом. Проработайте дополнительные загвоздки, в том числе проблемы взаимодействия команд между собой.

Этап 3. Развернуть применение Agile-методов во всей компании. Применяя опыт, полученный на этапах 1 и 2, внедрите использование Agile в оставшихся командах и структурах организации. Задействуйте членов команд из этапов 1 и 2 в качестве коучей для других команд.

Это логично, интуитивно понятно и, кажется, даже работает. Но здесь упущены некоторые важные составляющие, которые необходимы для успешного внедрения, а также ключевое различие между внедрением в первых командах и внедрением в крупных масштабах.

Модель изменений по принципу домино

Изменение организации — это обширная тема, исследователи долгое время ее изучают и пишут на эту тему труды. Профессор Гарвардского университета Джон Коттер говорит о восьми шагах, необходимых для успешного изменения организации, которые поделены на три этапа [Коттер, 2012]:

- Создание благоприятной обстановки для изменений.
- Вовлечение организации и наделение ее полномочиями.
- Реализация и поддержание изменений.

В начале XX века психолог Курт Левин представил подобную идею:

- Разморозка.
- Изменение.

- Заморозка.

Эти модели дают пищу для размышлений. Чтобы предусмотреть, какую поддержку понадобится оказать для успешного внедрения Agile, мне нравится использовать модель изменений, вдохновленную работой Тима Кностера, которую я называю «модель изменений по принципу домино».

Согласно модели изменений по принципу домино (Demand Chain Management, DCM), для успешного изменения организации требуются следующие составляющие:

- Bˆидение.
- Согласованность.
- Навыки.
- Ресурсы.
- Стимулы.
- План действий.

Если присутствуют все составляющие, изменение пройдет успешно. Если же что-то отсутствует, изменение не произойдет.

Можно представить ее как костяшку домино, которая должна быть на своем месте. Если какая-то костяшка отсутствует, изменение не случится (рис. 23.1).

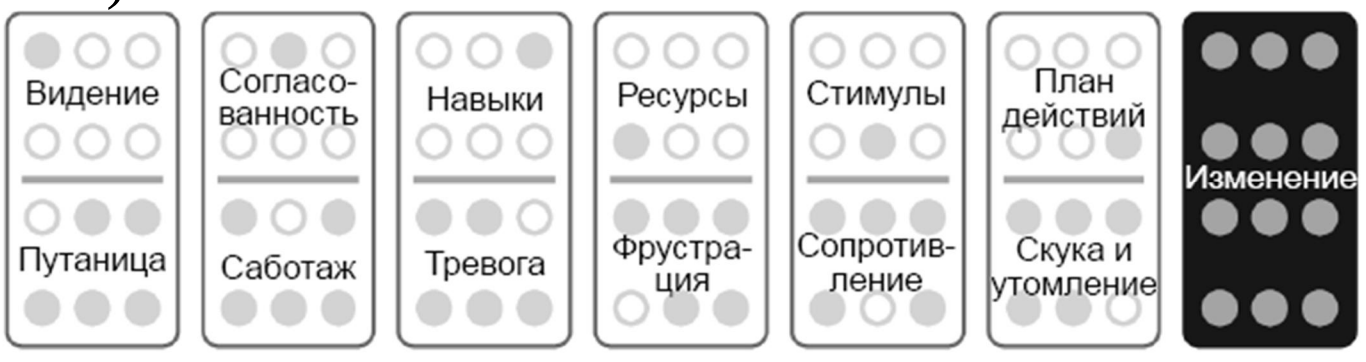


Рис. 23.1. Составляющие, необходимые для изменения, а также влияние их отсутствия согласно модели изменений по принципу домино

В оставшейся части этого раздела мы пройдемся по этим составляющим.

Видение

Согласно модели изменений по принципу домино (DCM), нехватка вˆидения приводит к путанице. Она начинается уже с самого определения Agile. Как я писал в главе 2, у разных людей может быть совершенно разное представление о том, что значит Agile. Без ясности вˆидения кто-то будет считать, что «внедрение Agile» означает перекройку всего предприятия для большей гибкости. А кто-то будет думать, что это просто реализация Скрама в компании целиком. Лидеры должны донести подчиненным четкое определение Agile.

Помимо такого определения вˆидение должно включать подробное объяснение желаемого конечного состояния. Во время такого объяснения нужно рассказать, зачем нужно внедрять Agile, какие преимущества это

привнесет, насколько широким будет внедрение и как оно повлияет на каждого сотрудника лично — лучше всего объяснить влияние внедрения не на категорию или группу, а на каждого из специалистов.

Продвижение изменений без ясности вˆидения приведет к ощущению того, что «руководство само не знает, чем занимается».

Согласованность

По модели DCM нехватка согласованности ведет к саботажу и диверсиям, нашей компании довелось быть свидетелем многочисленных примеров. Саботаж может быть разным — например, «каскадный Скрам» (использование каскадной модели с переименованием практических методов на манер Скрама), «Скрам, но» (опущение необходимых составляющих Скрама), преодоление малейших трудностей спустя рукава, недовольство и пассивное сопротивление.

Если руководство вводит изменения без обеспечения согласованности, то это приводит к ощущению того, что «руководству на нас плевать». Объяснение четкого вˆидения значительно способствует достижению согласованности, а такое активное взаимодействие даже не просто нужно — оно необходимо. Четкое объяснение преимуществ — это один из кратчайших путей внедрить Agile: команды решают, что это как раз то, что им нужно для успешной работы.

Достижение настоящей согласованности предполагает двустороннее взаимодействие: руководители излагают вˆидение и чутко реагируют на обратную связь о своем вˆидении. В ходе достижения настоящей согласованности можно повлиять на вˆидение. Руководителю нужно быть открытым для возможности изменения вˆидения, что является лишь еще одним примером принципа «изучайте и приспосабливайтесь».

Навыки

Вы не можете заставить сделать кого-либо что-то, что он не способен сделать, и поэтому попытка внедрить Agile без выработки нужных навыков приводит к тревожности. Когда руководство пытается продвигать изменения без развития необходимых навыков, создается ощущение, что «руководство сошло с ума». Выработка навыков требует основательного профессионального развития, в том числе формальной подготовки очно или дистанционно, дискуссионных групп, семинаров, времени для практики новых методов, внутреннего коучинга, внешнего коучинга и наставничества.

Ресурсы

Одна из распространенных тенденций заключается в том, что руководство хочет внести изменение и интересуется, почему оно занимает так много

времени, в то время как сотрудники также хотят внести это изменение, но уверены, что руководство не позволит им это осуществить. Мы называем это обостренными отношениями между руководством и сотрудниками.

Одна из причин заключается в том, что руководство просит сотрудников внести изменение без необходимых на то ресурсов — тогда они обязательно станут чувствовать, будто бы им не дают возможности внести это изменение.

Учитывая, что разработка ПО — это искусная интеллектуальная работа, ресурсы, необходимые для изменения процесса разработки, включают в себя доступ к обучению, коучингу и лицензиям на инструменты. Хотя может казаться, что в этом нет необходимости, сотрудникам также необходимо явное разрешение на выполнение работы по внедрению и явно выделенное на это время. Без этого в приоритете будут находиться повседневные задачи. Крупным организациям обычно нужны штатные специалисты для внедрения.

Без достаточных ресурсов персонал будет ощущать, что «руководство на самом деле не это имеет в виду».

Стимулы

Без стимулов вероятно сопротивление. Это естественно, потому что людям не хочется изменений, в которых нет их личного интереса. Большинство людей считают, что в их интересах сохранить комфорт текущего положения дел — любое изменение требует обоснования.

Это еще одна область, где помогает внятно выраженное вˆидение. Стимулы не обязательно должны быть денежными и даже не обязательно осязаемыми. Каждому человеку нужно понимать, почему изменение для него важно, в чем заключается его личный интерес. Это большая работа, которая требует большого количества устойчивого взаимодействия. Но без нее персонал будет ощущать, что «руководство пользуется нами».

Не забывайте учитывать важность автономии, профессионализма и цели. Высококачественная реализация Agile будет усиливать автономию команд и специалистов. Внимание, сосредоточенное на эмпирически основанном планировании, и мышление, ориентированное на рост, будут способствовать обучению и профессионализму. Наилучший стиль руководства для поддержки работы команд по Agile — регулярно доносить цель.

План действий

Без плана действий ваше внедрение заглохнет. Нужно давать задачи конкретным людям и устанавливать сроки. Планы нужно доносить до всех участников. Это просто, но об этом постоянно забывают: если люди не знают, как поспособствовать внедрению, они не станут этим заниматься!

Продавливание внедрения без плана действий ведет к ощущению, что «руководство не стремится к переменам».

В крупных организациях распространен паттерн — инициирование большого количества циклов изменений, большинство из которых никогда ни к чему не приведет. После нескольких таких циклов специалисты берут на вооружение подход — лечь на землю и надеяться, что взрывная волна изменений благополучно их минует, не поразив. Опыт их организации показывает, что у такого подхода масса достоинств.

Не забывайте включать принцип «изучайте и приспосабливайтесь» в план действий. Изменение должно происходить постепенно и включать в себя улучшение внедрения на основе регулярных ретроспектив и применения накопленного опыта.

В табл. 23.1 показано общее влияние отсутствия какой-либо составляющей и ее влияние на восприятие подчиненными начальства согласно модели DCM.

Таблица 23.1. Влияние отсутствия составляющих модели DCM

Отсутствует	Приводит к	Ощущается подчиненными как
В^идение	Путанице	Руководство само не знает, чем занимается
Согласованность	Саботажу	Руководству на нас плевать
Навыки	Тревожности	Руководство сошло с ума
Ресурсы	Разочарованию	Руководство на самом деле не это имеет в виду
Стимулы	Сопротивлению	Руководство пользуется нами
План действий	Скуке и утомлению	Руководство не стремится к переменам

Распространение изменений по всей организации

Модель изменений по принципу домино (DCM) полезна как при планировании внедрения Agile, так и при выяснении причин неудачного внедрения.

Есть и другая сторона внедрения, которая отсутствует в этой модели, но связана с проблемами того, как организации испытывают практические Agile-методы и как они продолжают разворачивать применение этих методов в более крупном масштабе.

В противоположность идеализированному описанию внедрения Agile, приведенному в начале этой главы, его внедрение во многих организациях скорее выглядит вот так:

- Организация решает внедрить Agile.
- Внедрение в первой пробной команде заканчивается успехом.
- Вторая или третья группа команд, которая вводит изменения, внедряет Agile с запинками или заканчивает провалом — команды с треском проваливают внедрение, отказываются от новых методов и возвращаются к старым или невозможно найти команду, которая согласится пойти по стопам пробной команды.

Почему так происходит? Вы, вероятно, знакомы с моделью Джеффри Мура «Преодоление пропасти», поскольку она применяется к выводу на рынок инновационных продуктов [Мур, 1991]. Я обнаружил, что тот же принцип применим к введению инноваций внутри организации.

Модель Мура основана на новаторской работе Эверетта Роджерса по теории диффузии инноваций [Роджерс, 1995]. Поскольку понятие «пропасть», употребленное Муром, не относится к этому рассуждению, я сосредоточу внимание на описании, которое дал Роджерс.

В модели Роджерса инновации перенимаются слева направо среди категорий пользователей, показанных на рис. 23.2.

Рис. 23.2. Последовательность внедрения инноваций

У каждой категории присутствуют определенные черты. *Новаторы*(самые первые внедряющие) предприимчивы и с радостью хотят пробовать новые технологии и методы. Их привлекают инновации сами по себе. Они способны справиться с высокой степенью неопределенности и хорошо переносят риски. Их часто постигают неудачи, но их это не расстраивает, потому что они высоко мотивированы перспективой быть первыми, кто привнесет в работу что-то новое. Из-за частых неудач их могут не уважать специалисты из других категорий.

Ранние последователи несколько похожи с новаторами, но их рвение не так высоко. Им также интересны новые технологии и методы, в основном из-за того, что они хотят создать что-то крутое раньше, чем это сделает кто-то другой. Ранние последователи не так часто терпят крах, как новаторы, поэтому их мнение весьма значимо в их структуре. Они являются хорошим примером для прочих внедряющих.

У новаторов и ранних последователей есть несколько общих черт. Обе категории интересуются инновациями самими по себе. Они занимаются поиском революционных, качественно новых достижений. Они устойчивы к риску и высоко мотивированы увидеть новые изменения в деле. Они готовы приложить значительное количество собственных усилий и проявить личную инициативу, для того чтобы изменения заработали. Они будут изучать материал, искать товарищей по интересам, экспериментировать и так далее. Они воспринимают трудности, связанные

с чем-то новым, как возможности довести что-то новое до рабочего состояния раньше остальных. Вывод заключается в том, что эти люди могут успешно справиться с внедрением без значительной поддержки.

Теперь главный вопрос: кто обычно работает над пробными командами?

Новаторы и ранние последователи, естественно! В этом-то и проблема, поскольку большинство сотрудников представлено не ими — они составляют довольно малую долю сотрудников.

Как показано на рис. 23.3, последовательность внедрения инноваций представляет собой стандартное нормальное распределение (гауссову кривую). Новаторы — это третье отклонение от медианы, а ранние последователи — второе. Вместе они составляют только 15% общего числа внедряющих.

Как и те, кто первыми пробуют новшества (новаторы и ранние последователи), поздние последователи (85%) имеют общие, противоположные первым, характеристики. Их привлекает новизна ради достижения лучшего уровня качества или производительности, но не ради инноваций самих по себе. Они ищут безопасные методы с низким риском и постепенными последствиями. Они не очень хорошо переносят риск, многие боятся его. Они далеки от желания тратить свои силы на преодоление препятствий, потому рассматривают препятствия как свидетельство того, что изменения вредны и от них лучше отказаться. Их не очень мотивирует увидеть успешную реализацию изменений — по большей части им безразлично, их мотивация колеблется, они то хотят успеха внедрения, то его провала.

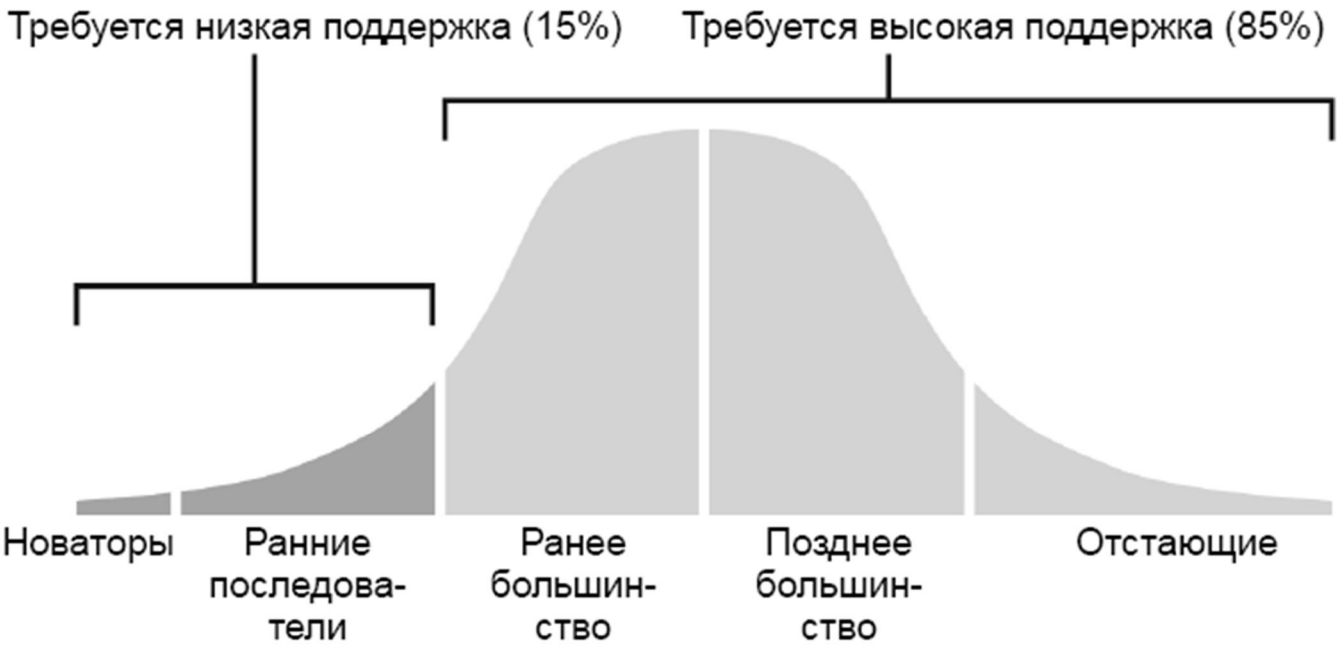


Рис. 23.3. Для разных частей последовательности внедрения инноваций требуется разный уровень поддержки. Поздние внедряющие требуют более высокого уровня поддержки, чем ранние

Это означает, что пробные команды не склонны говорить бˆольшую часть того, что вам нужно знать для успешного внедрения. Поздним последователям понадобится бˆольшая поддержка, а именно они составят большинство.

Некоторые руководители технических организаций станут спорить, что у них куда бˆольшая доля новаторов и ранних последователей и меньшая доля раннего большинства, позднего большинства и отстающих. На самом деле в разных коллективах доля может отличаться и этот взгляд может быть верным. Но внутри этих коллективов все равно будет значимый разрыв. Ранние последователи будут производить оценки, а поздним понадобится относительно больше поддержки.

Взгляд издалека на внедрение, дубль 2

А теперь взгляд издалека на более реалистичный подход к внедрению Agile:

Этап 1. Начать с пробной команды. Соберите первую команду для пробного внедрения Agile в компании. Проработайте возникающие трудности, работая с одной командой.

Этап 2. Распространить работу по Agile на одну или более команд дополнительно. Донесите в подробностях то, как Agile поможет компании и сотрудникам. Опишите в подробностях преимущества, полученные пробной командой. Расскажите о том, как внедрение Agile пойдет на пользу конкретным людям в следующих командах. Проведите открытые курсы в рабочее время и коучинг, выделите время на внедрение изменений в новых командах. Учредите сообщества практиков и поощряйте их. Регулярно сверяйтесь с новыми командами и предлагайте дополнительную поддержку. Проработайте дополнительные трудности, в том числе проблемы взаимодействия команд между собой. Разработайте план проведения обучения и обеспечения поддержки, необходимых для масштабного внедрения.

Этап 3. Развернуть применение практических Agile-методов во всей компании. Изучите опыт внедрения в первых нескольких командах и пересмотрите вˆидение. Донесите обновленное вˆидение того, как Agile поможет компании и сотрудникам. Опишите в подробностях преимущества, полученные теми командами, и объясните, какие уроки, которые гарантируют успех внедрения в новых командах, были извлечены. Прислушивайтесь к обратной связи и пересматривайте вˆидение по мере необходимости. Донесите обновленное вˆидение сотрудникам и сообщите, что вы учли обратную связь от них.

Запланируйте встречи с каждым сотрудником, на которого повлияет внедрение Agile, и обсудите подробное вˆидение того, какую пользу внедрение Agile принесет каждому. Подготовьтесь к каждой из этих встреч, предварительно изучив каждый отдельный случай. Не относитесь к членам коллектива как к безликим представителям какой-либо группы.

Опишите план успешного внедрения Agile в компании. Опишите, кто прилагает больше всего усилий к внедрению, какие задачи нужно будет

выполнить для успешного внедрения, а также временнˆую шкалу внедрения.

Проводите курсы и коучинг в рабочее время. Подчеркивайте то, что у каждой команды есть разрешение на выполнение работы, необходимой для успешного внедрения. Регулярно сверяйтесь с командами и оказывайте дополнительную поддержку. Выделите специалистов для решения проблем внутри команд и между командами. Объясните, что ожидаются трудности и что при их возникновении будет оказана поддержка.

В общем, проявляйте при внедрении решимость командира. Совместно придите к общему вˆидению, и пусть люди сами дальше прорабатывают подробности.

Изучайте и приспосабливайтесь

По мере внедрения периодически пересматривайте модель изменений по принципу домино (DCM) и обращайте внимание на признаки наличия проблем в каждой из областей. Каждое внедрение в каком-то смысле неповторимо. Будьте открыты для обратной связи и будьте готовы сменить вектор при необходимости. Это возможность смоделировать принцип «изучайте и приспосабливайтесь» на уровне руководства.

Рекомендации

Изучайте

- Пересмотрите модель изменений по принципу домино (DCM) и то, как ее можно применить к вашим начинаниям по внедрению изменений в настоящем и прошлом. В каких частях модели ваша компания преуспевает? Есть ли куда расти?
- Подумайте над моделью диффузии инноваций и над тем, как она применима к опыту, который компания получила от пробных команд. Согласны ли вы с тем, что ваши пробные команды состоят (или состояли) из инноваторов или ранних последователей? Как они были представлены в другой части компании?

Приспосабливайтесь

- На основе сравнительного анализа текущего внедрения Agile и модели изменений по принципу домино создайте план улучшения производительности там, где она проседает.
- На основе сравнительного анализа вашей текущей поддержки поздних внедряющих и модели диффузии инноваций создайте план обеспечения соответствующей поддержки.

Дополнительные ресурсы

[Эверетт М. Роджерс, 1995]. Diffusion of Innovation, 4th Ed. Это исчерпывающая работа по диффузии инноваций.

[Джеффри Мур, 1991]. Crossing the Chasm, Revised Ed. Эта книга популяризировала работу Роджерса по диффузии инноваций. Она отлично читается и гораздо короче, чем книга Роджерса.

[Рональд А. Хейфец и Марти Линski, 2017]. Leadership on the Line: Staying Alive Through the Dangers of Change, Revised Ed. Это строгая книга, которая, тем не менее, предлагает действительно полезный способ подумать о роли руководителя во внедрении основных изменений (вид со стороны) и некоторых важных препятствиях изменениям, которые редко обсуждают.

[Джон П. Коттер, 2012]. Leading Change. Исчерпывающая работа Коттера по внедрению основных изменений. Джон Коттер и Холгер Ратгебер. 2017. Our Iceberg Is Melting, 10th Anniversary Edition. Это развлекательная версия теории изменений Коттера, которую донесли в форме притчи о пингвинах. Если вам нравятся книги в стиле «Who Moved My Cheese and Fish!» и «Рыба! Проверенный способ поднять моральный дух и улучшить результаты», то понравится и эта.

[Кори Мадстен, 2016]. How to Play Dominoes. Каждый автор становится чуть напористее к концу написания своей книги. Видели, что я только что написал? Значит, вы еще читаете эту книгу.

Часть V. Заключение

В этой части изложен взгляд на правильное функционирование организации по Agile, а также подытожены ключевые принципы, которые приводились на протяжении книги.

Наслаждайтесь плодами своих трудов

С самого своего начала Agile служил и боевым кличем, объединяющим сотрудников для создания ПО лучшего качества, и общим понятием, объединяющим огромное количество различных практических методов, принципов и взглядов, которые продолжают свое развитие в поддержку того самого боевого клича.

Agile сам по себе следует принципу «изучайте и приспосабливайтесь», чтобы стать лучше, — именно поэтому сейчас Agile лучше, чем тот, который мы знавали двадцать лет назад. В современном Agile есть понимание, что его цель — не просто соблюдать какие-то каноны. Цель Agile заключается в применении практических методов в поддержку целей и стратегий вашего бизнеса.

Эффективный Agile начинается с лидера — то есть вы задаете темп командам. Четко донесите ожидания с решимостью командира, наделите команды полномочиями, развивайте у них способности к самоуправлению, а затем позвольте им работать итерациями и совершенствовать свои навыки. Меняйте систему и процессы, а не людей. Помогайте компании декриминализировать ошибки и развивать мировоззрение, ориентированное на рост. Пользуйтесь ошибками как возможностью для

обучения, применения принципа «изучайте и приспосабливайтесь» и постепенно совершенствуйтесь.

Когда вы научитесь это хорошо делать, компания будет создавать команды, сосредоточенные на целях компании. Команды будут чутко и оперативно реагировать на потребности компании, даже когда они изменяются. Это улучшит способность вашей организации реагировать на изменение потребностей своих клиентов.

Команды станут отслеживать эффективность используемых методов и заменять неэффективные лучшими методами. Со временем их пропускная способность увеличится.

Команды будут постоянно отслеживать свои рабочие потоки. Они будут знать, на какой стадии работа и идет ли она согласно плану. Они будут предоставлять полный доступ для остальных команд. Они будут доставлять именно то, о чем был разговор, — непременно с высоким качеством работ.

Команды будут работать сплоченно — с другими командами, с другими стейкхолдерами и со всем миром, окружающим вашу компанию.

Открытия будут происходить постоянно, но разрушительные неожиданности будут редкостью. В случае таких неожиданностей команды будут вовремя осведомлять о них, что позволит оперативно и эффективно реагировать как командам, так и компании в целом.

Команды будут всегда поддерживать высокий уровень качества и находить возможности для роста. Мотивация будет на высоте, а издержки невелики.

По мере того как компания будет продвигаться к такому взгляду на эффективность разработки ПО, ей придется пройти через несколько этапов созревания.

Изначально основное внимание будет сосредоточено на производительности внутри команды. Командам понадобится несколько спринтов, чтобы постичь Скрам и вспомогательные Agile-методы. Они будут работать над умением планировать работу пошагово небольшими инкрементами, проектировать способами, которые позволяют работать короткими итерациями, выставлять приоритеты, вносить код, поддерживать высокий уровень качества, принимать решения от лица организации, работать в команде и доставлять функционал. В зависимости от того, насколько хорошо им оказывают поддержку, и того, как много помех им встречается со стороны организации, им может понадобиться много спринтов, чтобы достичь высокого уровня своих способностей.

С течением времени внимание сместится на взаимодействие между компаниями и командами. Поскольку производительность команд выросла, компании понадобится оказывать им поддержку в виде четкого руководства в отношении продукта — в отношении приоритетов

требований и прочих приоритетов, а также в виде своевременного принятия решений, которое соответствует возросшему темпу производительности команд.

В конце концов, изменения в сторону итеративности преобразят ваши команды. Они будут быстро доставлять функционал и смогут быстро менять направление. Это откроет стратегические возможности для вашей организации, что позволит планировать и выполнять планы иначе, лучше, благодаря тому что ваши способности к разработке выросли.

Внимание, сосредоточенное на мировоззрении, ориентированном на рост, а также на принципе «изучайте и приспосабливайтесь», означает, что со временем все будет идти только к лучшему.

Наслаждайтесь результатами!

Краткое изложение ключевых принципов

Изучайте и приспосабливайтесь. Agile представляет собой эмпирический подход, который зависит от способности учиться исходя из опыта. Для этого требуется создавать возможности для периодического переосмысления и внесения правок в процесс, основываясь на опыте (с. 35).

Начните со Скрама. Скрам не обязательно является конечной точкой в путешествии по Agile, но он лучше всего структурирован и лучше всего подходит для того, чтобы с него начать (с. 39).

Обеспечьте кросс-функциональность команд. При выполнении проекта по Agile работа над ним ведется в самоуправляемых командах. Чтобы быть самоуправляемыми, командам нужно обладать полным набором навыков, которые необходимы для принятия обоснованных, обязательных для организации решений (с. 59).

Интегрируйте тестировщиков в команды разработчиков. Обеспечьте тесную обратную связь между разработчиками и тестировщиками посредством обеспечения более тесной совместной работы специалистов (с. 63).

Мотивируйте команды автономией, профессионализмом и целями. Практические Agile-методы в своей основе способствуют укреплению мотивации. Команды нацелены на автономную работу и совершенствование со временем (профессионализм). Чтобы так и было, им нужно понимать свою цель. Понятие «здоровой команды» в Agile тесно переплетено с понятием «мотивированной команды» (с. 69).

Развивайте мировоззрение, ориентированное на рост. Хоть смотрите на это с точки зрения профессионализма из троицы автономии, профессионализма и цели, хоть с точки зрения принципа «изучайте и приспосабливайтесь» — команды, работающие по Agile эффективно, постоянно стремятся к совершенству (с. 72).

*Развивайте ориентацию на бизнес.*Разработчикам зачастую нужно заполнять пробелы в требованиях и в указаниях владельца продукта. Понимание

бизнеса поможет им заполнить эти пробелы так, чтобы бизнесу это пошло на пользу (с. 75).

Обеспечьте более тесную обратную связь. Не тратьте времени на усвоение уроков больше, чем на то требуется. Поддерживайте как можно более тесную обратную связь. Это способствует скорейшему продвижению хода работ благодаря принципу «изучайте и приспосабливайтесь», а также скорейшему росту эффективности благодаря развитию мировоззрения, ориентированного на рост (с. 78).

*Меняйте систему, а не людей.*Большинство профессионалов в области разработки ПО желают работать хорошо. Если они работают плохо, в особенности если кажется, что они вообще не стараются, — разберитесь, почему так происходит и что к этому приводит. Ищите в системе те проблемы, которые снижают моральный дух людей (с. 85).

Развивайте способности команды посредством развития людей. Команды обладают теми свойствами, которые складываются из индивидуальных свойств членов команды и взаимодействия между ними. Укрепляйте способности ваших команд посредством развития специалистов по отдельности (с. 90).

*Небольшие размеры проектов.*Небольшие проекты легче выполнять, они чаще завершаются успехом. Не всю работу можно разбить на малые проекты, но если ее можно распределить по небольшим проектам, то лучше так и сделать (с. 102).

Короткие спринты. Спринты небольшой длительности способствуют частому обновлению обратной связи из принципа «изучайте и приспосабливайтесь». Проблемы быстро выходят наружу — становится легче пресекать маленькие проблемы на корню, не давая им разрастись (с. 105).

Доставка продукта вертикальными срезами. В Agile важна обратная связь. Команды получают улучшенную обратную связь в отношении технологических и проектировочных решений как от клиентов, так и от бизнеса, если они выполняют доставку вертикальными, а не горизонтальными срезами (с. 110).

*Управляйте техническим долгом.*Внимание, устойчиво сосредоточенное на качестве, — часть эффективной реализации Agile. Управление техническим долгом поддерживает повышенный уровень морального духа в команде, скорый прогресс и высокое качество продуктов (с. 112).

Поддержка крупных Agile-проектов с помощью архитектуры. Добротная архитектура может позволить распределить работу над проектом и свести к минимуму издержки, возникающие из-за большого размера проекта. Качественно построенная архитектура может создать ощущение того, что вы работаете не над крупным проектом (с. 123).

Быстрее находите дефекты. Как показывает практика, чем дольше недочет оставляют без внимания, тем дороже потом его исправление.

Преимущество Agile в том, что больше недочетов удается обнаружить как можно скорее. Это возможно благодаря тому, что постоянное внимание уделяется работе над качеством (с. 132).

Создайте и применяйте критерии готовности. Правильно сформулированные критерии готовности помогают как можно раньше выявить работу, выполненную не полностью или неправильно. Это максимально сокращает время между появлением и обнаружением недочета (с. 134).

Поддерживайте качество на уровне релиза. Поддержание качества на уровне релиза помогает найти недочеты, которым удалось проскользнуть через критерии готовности (с. 136).

*Применяйте автотесты, созданные командой разработчиков.*Автоматизированные тесты помогают максимально сократить время обнаружения недочета. Назначение каждого члена команды ответственным за тестирование укрепляет идею, что за качество работ отвечают все (с. 140).

*Уточняйте бэклог продукта.*Благодаря уточнению бэклога команда работает над задачами с наивысшим приоритетом, ей не приходится самостоятельно заполнять пробелы в требованиях и испытывать недостатки в работе (с. 156).

Создайте и применяйте критерии готовности задачи. Частично уточнение бэклога заключается в том, чтобы требования были действительно готовы, прежде чем команда начнет их реализацию (с. 157).

Автоматизируйте повторяющуюся деятельность. Никому не нравится однообразная деятельность, причем многие действия, которые возможно автоматизировать при разработке ПО, приносят больше пользы, если их таки автоматизировать (с. 170).

Управляйте результатами, а не мелочами. Поддерживайте автономию ваших команд, ясно донося желаемые результаты, в то же время оставляя команде свободу выбора в том, как именно она будет выполнять свою работу (с. 178).

Выражайте цель с решимостью командира. Поддерживайте у команд способность своевременно и самостоятельно принимать решения — для этого нужно ясно доводить до них цели, соответствующие желаемому конечному состоянию (с. 179).

Сосредоточьтесь на пропускной способности, а не на деятельности. Так же как с результатами, принимая во внимание тот нюанс, что загруженность работой и бурная деятельность не является целью. Целью является выполнение работы, представляющей ценность (с. 181).

*Моделируйте ключевые стороны поведения при работе по Agile.*Эффективные руководители показывают поведение, которое хотят видеть у остальных (с. 182).

*Декриминализируйте ошибки.*Декриминализируйте ошибки, чтобы команды не боялись и не скрывали их, чтобы у вас была возможность на них

научиться. За ошибки, из которых не усвоено никаких уроков, компания заплатит дважды (с. 184).

Стройте планы, основываясь на измеряемых возможностях команды. Agile представляет собой эмпирический подход — команды и организации должны планировать свою работу на основе показателей, полученных в результате измерения их способностей (с. 187).

Благодарности

Прежде всего хочу поблагодарить своих коллег из *Construx Software*. Мне посчастливилось работать с необыкновенно умными, талантливыми и опытными специалистами, ведь эта книга преимущественно описывает наш совместный опыт, поэтому я не смог бы ее написать без этих людей. Спасибо Дженни Стюарт, вице-президенту по консалтингу, за ее невероятный опыт и выводы, полученные из наблюдений при масштабных внедрениях Agile. Я ценю ее комментарии в отношении проблем внедрения в крупных организациях.

Спасибо Мэтту Пелокину, техническому директору, за его компетенцию, которая не имеет себе равных во всем мире, в области архитектуры ПО (более 500 обзоров), а также за понимание той роли, которую архитектура играет в реализации Agile. Спасибо Эрлу Биду, старшему научному сотруднику, консультанту и экстраординарному коучу, за его понимание способов, позволяющих настолько доходчиво преподнести концепции Agile, чтобы команды смогли их понять и эффективно претворять в жизнь. Спасибо Мелвину Пересу Седано, старшему научному сотруднику, за сочетание глубокой начитанности и знания мирового опыта. Спасибо Мелвину за то, что в течение всего проекта он был настоящей ходячей энциклопедией и потрясающе ориентировал меня в наиболее эффективных практических методах. Спасибо Эрику Симмонсу, старшему научному сотруднику, — он просто неисчерпаемый источник знаний в области исследований неопределенности и сложности, он также обладает экспертными знаниями о реализации практических Agile-методов в крупных традиционных компаниях. Спасибо Стиву Токи, главному консультанту, за его глубокое понимание и бесподобные фундаментальные знания о традиционных, жестких практических методах разработки ПО и о том, как они взаимодействуют с методами Agile. Спасибо Бобу Вебберу, старшему научному сотруднику, за его понимание управления продуктами в Agile — его многолетний опыт руководителя помог сосредоточить основное внимание этой книги на том, что нужно руководителям. И наконец, спасибо Джону Клиффорду, ведущему руководителю по методам Agile, за его богатый опыт в поощрении, коучинге, обучении и за то, что от случая к случаю добивался от организаций понимания всей той ценности, которую

можно обрести благодаря внедрению Agile. Столько талантов в одном месте! Как же мне повезло работать с такими людьми!

Более 300 руководителей в области разработки ПО прочли первые черновики этой книги, оставив к ней комментарии и предложив свои правки. Книга стала несоизмеримо лучше благодаря их щедрому вкладу.

Особая благодарность Крису Адександеру за подробное объяснение петли Бойда и потрясающие примеры. Особая благодарность Берни Энджеру за его развернутый комментарий о том, как преуспеть в роли владельца продукта. Особая благодарность Джону Белбуте за его ценные замечания по поводу измерений и улучшения процесса. Особая благодарность Биллу Кертису и Майку Расселу за то, что дали мне нагоняй за некоторые заблуждения в отношении принципа Деминга — Шухарта (PDCA), которые я в итоге исключил из книги. Особая благодарность Робу Дагено за его замечания об архитектуре, непрерывных интеграции и доставки.

Большая благодарность Брайану Дональдсону за оценку глубины текста. Особая благодарность Ларсу Маровски-Бри и Эду Салливану за исчерпывающие комментарии по факторам, необходимым для успешной работы с географически распределенными командами. Особая благодарность Мариону Миллеру за описание того, как организованы поисково-спасательные команды и как это относится к организациям, работающим по Agile. И отдельная благодарность Брайану Пфлугу за обширный комментарий о регулирующих нормах разработки ПО в аэрокосмической отрасли.

Я также ценю помощь рецензентов, которые оставляли комментарии к отдельным частям проекта. Ниже я перечислю их: Марк Абермоске, Анант Адке, Хайтам Альцейни, Прашант Амбе, Видиха Ананд, Ройс Осберн, Джозеф Балистриери, Эрика Барбер, Эд Бейтман, Марк Бердалл, Грег Бертони, Диана Биттл, Маргарет Бон, Терри Бретц, Дарвин Кастильо, Джейсон Коул, Дженсон Кроуфорд, Брюс Кронквист, Питер Дейли, Брайан Догерти, Мэтт Дэйви, Пол Дэвид, Тим Доусон, Ритеш Десаи, Энтони Диаз, Рэнди Дойутрек, Адам Дрей, Эрик Эванс, Рон Фаррингтон, Клаудио Фаяд, Джефф Фламанк, Лиза Форсайт, Джим Форсайт, Робин Франко, Джейн Фрейзер, Фазил Гарибу, Инбар Газит, Дэвид Гевинг, Пол Гауэр, Ашиш Гупта, Крис Хэлтон, Рам Харихаран, Джейсон Хиллс, Гари Хинкль, Майк Хоффман, Крис Холл, Питер Хорадан, Сандра Хоулетт, Фред Хуганд, Скотт Дженсен, Стив Кармезин, Питер Крецман, Дэвид Лейб, Эндрю Левин, Эндрю Личей, Эрик Линд, Ховард Лук, Чжи Конг (Чонг) Ло, Дейл Лутц, Марианна Марк, Кит Б. Маркос, Дэвид Мактавиш, Джей Ди Мейер, Сунел Мендиратта, Генри Море, Бертран Мейер, Роб Мьюр, Крис Мерфи, Пит Натан, Майкл Нассириан, Скотт Нортон, Даниэль Ренсилот Окин, Ганеш Палаве, Питер Пазнокас, Джим Пайлс, Марк Ронан, Рошанак

Рошандель, Хиранья Самарасекера, Налин Савара, Том Шафферно, Сенти Сентилмуруган, Чарльз Сейбольд, Эндрю Синклер, Том Спитцер, Дэйв Спокан, Майкл Сприк, Тина Стрэнд, Нэнси Сазерленд, Джейсон Таннер, Крис Томпсон, Брюс Торн, Линн Треворроу, Роджер Велейд, Джон Уорд, Венди Уэллс, Гавиан Вишоу и Говард Ву.

Спасибо рецензентам, которые оставляли комментарии ко всей рукописи. Вот они: Эдвин Адриаансен, Карлос Амселем, Джон Андерсон, Мехди Ауади, Марк Апгар, Брэд Эпплтон, Джованни Аспрони, Джозеф Бэнгс, Алекс Баррос, Джаред Беллоус, Джон М. Бемис, Роберт Биндер, Майк Блэксток, доктор Зарик Богоссян, Габриэль Бойчук, Грег Борхерс, Ксандер Бота, Мелвин Брэндман, Кевин Брюн, Тимоти Бирн, Дейл Кэмпбелл, Майк Каргал, Марк Кэссиди, Майк Ченг, Джордж Чоу, Ронда Силсик, Питер Кларк, Мишель К. Коул, Джон Коннолли, Сара Купер, Джон Костер, Алан Крауч, Джеймс Кьюсик, Дэвид Дейли, Трент Дэвис, Дэн Делапп, Стив Динстбир, Ильхан Дилбер, Николас Дилизи, Джейсон Домас, Дэвид Драффин, доктор Райан Дж. Дюранте, Джим Даррелл, Алекс Элентух, Пол Элиа, Роберт А. Энсинк, Эрл Эверетт, Марк Фамос, Крейг Фишер, Джейми Форган, Иэн Форсайт, Джон Р. Фокс, Стивен Д. Фрейзер, Стив Фриман, Рив Фричман, Кристиан Гаспар, Мэнни Гатлин, Ридже Джордж, Гленн Гудрич, Ли Грант, Кирк Грей, Мэтью Грульке, Мир Хаджмирага, Мэтт Холл, Колин Хаммонд, Джефф Хансон, Пол Хардинг, Джошуа Хармон, Грэм Хейторнтуэйт, Джим Хенли, Нед Хенсон, Нил Херман, Сэмюэль Хон, Дьюи Хоу, Билл Хамфри, Лиз Хватум, Натан Ицкович, Роб Джаспер, Кури Джеймс Джадд, Марк Карен, Том Керр, Йогеш Хамбия, Тимо Киссел, Кэти Ноббс, Марк Кочански, Ханну Кокко, Сунил Крипалани, Мукеш Кумар, Сумант Кумар, Мэтт Кузницки, Стефан Ландвогт, Майкл Ланге, Эндрю Лаверс, Роберт Ли, Энтони Гилберт Левеск, Рон Лихти, Кен Лю, Джон Лофтен, Серхио Лопес, Арни Лунд, Джефф Малек, Коэн Маннертс, Ристо Матикайнен, Крис Мэттс, Кевин Макиерн, Эрнст Менет, Карл Мативье, Скотт Миллер, Правин Минуре Дэвид, Бра Мур, Шон Морли, Стивен Маллинс, Бен Нгуен, Райан Норт, Луи Ормонд, Патрик О'Рурк, Ума Палепу, Стив Перрен, Даниэль Петерсен, Брэд Портер, Терри Поттс, Джон Прайс, Джон Пурди, Младен Радович, Венкат Рамамурти, Вину Рамасами, Дерек Рединг, Барбара Роббинс, Тим Роден, Нил Рудин, Деннис Рубсам, Джон Сантамария, Пабло Сантос Луасес, Барри Сэйлор, Мэтт Схаутен, Дэн Шрайбер, Джефф Шредер, Джон Селларс, Дон Шафер, Деш Шарма, Дэвид Шолан, Крейг Р. Смит, Дейв Смит, Хоуи Смит, Стив Снайдер, Митч Соннен, Эрик Сова, Себастьян Спек, Курк Спендлов, Тим Штауффер, Крис Стерлинг, Питер Стивенс, Лорейн Стейн, Йоаким Сунден, Кевин Тейлор, Марк Тристан, Билл Такер, Скот Туткович, доктор Кристиан П. Валке, Пол ван Хаген, Марк Х. Уолдрон, Боб Вамбах, Эван Ван, Фил Уайт, Тим Уайт, Джон Уитни,

Мэтью Уиллис, Боб Уилмс, Дэвид Вуд, Ронни Йейтс, Том Йосик и Барри Янг.

Среди множества комментариев некоторые выделялись особенной глубиной и полезностью. Особая благодарность за них Джону Аукшунасу, Сантану Банерджи, Джиму Берду, Аластеру Блейки, Мишель Кэнфилд, Геру Клаудту, Терри Коатте, Чарльзу Дэвису, Робу Дуллу, Рику Эссениусу, Райану Флемингу, Тому Грину, Оуэну Гриффитсу, Крису Хаверкате, доктору Арне Хоффманну, Брэди Хонсингеру, Филиппу Крухтену, Стиву Лэйну, Эшлин Лихи, Камилу Литману, Стиву Мараспину, Джейсону Маккартни, Майку Мортону, Шахиде Низар, Эндрю Парку, Джемми Пэйту, Джону Рейндерсу, Андре Синтзоффу, Питу Стюнтцу, Барбаре Тэлли, Эрику Апчерчу, Максасу Володину, Райланду Уоллесу, Мэтту Уорнеру, Уэйну Уошберну и Дэвиду Уайту.

Также хочу отметить отличную работу нашей производственной команды, в том числе Роба Нэнса за работу с графикой, Тоню Римби за руководство нашей обзорной инициативой и Джоан Спротт за составление указателей. Также спасибо Джесси Бронсону, Полу Доновану, Джеффу Элерсу, Мелиссе Феро, Марку Гриффину и Марку Нигрену за набор рецензентов.

Наконец, суперблагодарность Девону Масгрейву, редактору проекта. Это мой третий совместный с Девоном проект. Его редакторские решения во многом приблизили эту книгу к совершенству, а его неугасаемый интерес сыграл важную роль в том, чтобы эта книга появилась на свет.

Об авторе

Стив Макконнелл прежде всего известен как автор книги «Совершенный код», ставшей классикой в области разработки ПО. О ней часто говорят как о самой проработанной и продаваемой книге в этой отрасли. Книги Стива переведены на 20 языков и продаются миллионными тиражами по всему миру.

Компания Стива *Construx Software*вот уже более 20 лет помогает компаниям, занимающимся разработкой, совершенствовать свои навыки. Философия *Construx*заключается в том, чтобы обеспечить успех каждого проекта по разработке за счет повышения профессиональной эффективности людей, команд и организаций.

1 Стандарт MIL-STD-2167A, предполагающий каскадную модель разработки программного обеспечения в проектах Министерства обороны США, в конце 1994 года заменили на стандарт MIL-STD-498, предполагающий никак не связанную с каскадной модель разработки.

2 Пинк Дэниел. Драйв. Что на самом деле нас мотивирует. М.: Альпина Паблишер, 2013.

3 Точная формулировка закона Конвея такова: «Организации, проектирующие системы, ограничены дизайном, который копирует структуру коммуникаций в этой организации».

4 DISC — четырехсекторная поведенческая модель для исследования поведения людей в окружающей их среде или в определённой ситуации.

5 Брукс Фредерик. Мифический человеко-месяц, или Как создаются программные системы. СПб.: Питер, 2021. 368 с.: ил.

6 Однажды я работал с одним исполнительным директором, который видел своего начальника лишь полчаса раз в полгода.

7 Доверительный интервал — это особый (и сложный) статистический расчет того, насколько вы можете быть уверены в том, что наблюдаемое среднее значение близко к фактическому. В этом примере «доверительный интервал с вероятностью 90%» означает, что вы можете быть на 90% уверены, что фактическое среднее значение скорости будет находиться между 44 и 56, предполагая, что количество спринтов составит от 22 до 27 (принимая во внимание 4 спринта, которые уже завершены). Некоторые команды используют стандартное отклонение для расчета возможных результатов, но это математически некорректно. Стандартное отклонение позволяет предсказать значения скорости в отдельных спринтах, которые попадают в диапазон. Доверительный интервал — подходящий метод для расчета вероятного диапазона средней скорости для всех спринтов проекта.

Переводчик *И. Сигайлюк*

Художник *В. Мостипан*

Корректоры *М. Молчанова (Котова), М. Одинокова*

Стив Макконнелл

Еще более эффективный Agile. — СПб.: Питер, 2021.

ISBN 978-5-4461-1705-5