МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Омский государственный технический университет»

А. Г. Белик, В. Н. Цыганенко

КАЧЕСТВО И НАДЕЖНОСТЬ ПРОГРАММНЫХ СИСТЕМ

Учебное пособие

Омск
Издательство ОмГТУ
2018

УДК 004.4(075)
ББК 32.97я73
Б43

Рецензенты:
С.Н. Чуканов, д-р техн. наук, профессор,
зав. кафедрой «Компьютерные информационные автоматизированные системы» ФГБОУ ВПО «Сибирская государственная автомобильно – дорожная академия (СибАДИ)»;
П.Н. Надточий, канд. физ.-мат. наук,
зав.сектором разработки ПО ООО «Автоматика-Э»

Белик, А. Г.
Б43 Качество и надежность программных систем : учеб. пособие / А. Г. Белик, В. Н. Цыганенко ; Минобрнауки России, ОмГТУ. – Омск : Изд – во ОмГТУ, 2018. – 80 с. : ил.

На основе обзора нормативно – технической документации и литературных источников изложены методические основы определения, анализа и обеспечения показателей качества программных систем, приведена методика экспертной оценки качества программных систем различного назначения, рассмотрены способы расчета, моделирования и прогнозирования надежности программно-аппаратных комплексов автоматизированных систем обработки информации и управления.

Может быть использовано студентами следующих направлений бакалавриата: 09.03.01 «Информатика и вычислительная техника», 27.03.03 «Системный анализ и управление», 09.03.04 «Программная инженерия», магистратуры: 09.04.01 «Информатика и вычислительная техника».

УДК 004.4(075)
ББК 32.97я73

Печатается по решению редакционно – издательского совета Омского государственного технического университета

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Сложность программных систем, которые стимулируют эволюцию информационных технологий, растет быстрее, чем способность людей к ней адаптироваться. Это вызвало необходимость того, что ведущие отрасли перенесли свое внимание на процесс создания программного обеспечения, который превратился из индивидуального творчества в строгую дисциплину программной инженерии.

Программная интеграция наиболее активно стала проявляться с появлением Интернета и его огромными темпами роста, благодаря развитию средств взаимодействия. Компонентные платформы и открытые стандарты еще больше усиливают эту тенденцию. Успешное внедрение и интеграция отнюдь не тривиальны – чтобы предложить что-то полезное инженерам, новые технологии, процессы и средства инженерии нуждаются в аппарате глубокого управления изменениями.

Любая программная система должна выполнять определенные функции, при этом она должна иметь целый рядом свойств, позволяющий эффективно и успешно применять ее в течение длительного периода. Таким образом, она должна обладать требуемым уровнем качества, под которым понимается совокупность характеристик, определяющих способность удовлетворять заданным потребностям пользователей и автоматизируемых процессов, использующих данное программное средство [1,2]. Однако это не означает, что программные продукты должны обладать заданной совокупностью таких свойств в их наивысшей степени, так как повышение качества, как правило, можно осуществить только ценой изменения стоимости, увеличения сроков окончания проектирования и снижения других характеристик.

Проектная организация, занимающаяся разработкой программных систем, создает и реализует задачи поставки программ надлежащего качества, которые отвечают требованиям заказчиков и потребителей, а также укладываются в заданные временные и финансовые границы. В настоящее время для индустрия программного обеспечения свойственна высокаястепень конкуренции, что определяет выпуск качественных программ как одно из ключевых условий, обеспечивающих конкурентоспособность такой компании.

Быстрое увеличение сложности, величины и стоимости современных программных продуктов, а также ответственности реализуемых ими функций существенно повысило требования пользователей к эффективности и надежности его применения. Поэтому одним из основных критериев оценки качества программного обеспечения является степень его соответствия потребностям и ожиданиям пользователей.

При обеспечении качества программных систем одной из наиболее приоритетных задач является формализация т стандартизация показателей качества и эффективные методологии их оценки. Это означает, что при подготовке технических заданий на разработку программных систем необходимо четко прописываются основные требования к характеристикам их качества с указанием, методов их измерения или экспертной оценки для того чтобы обеспечить наибольшую достоверность результатов тестирования и надлежащее сравнение с требованиями, заданными в контракте и спецификациях. Это позволит сократить вероятность конфликтов между заказчиками и пользователями с одной стороны и разработчиками и поставщиками с другой вследствие разной интерпретации одних и тех же показателей качества.

Подобные противоречия могут быть устранены только на основе знания и правильного применения международных стандартов, регламентирующих жизненный цикл и методологию оценки качества программных систем. При этом следование этим стандартам не должно быть чисто механическим, так как необходима их адаптация к специфике и условиям применения конкретного программного проекта.

В современных условиях широкого применения инновационных информационных технологий проектирование программных систем превратилось в одну из наиболее дорогостоящих индустриальных технологий. Это означает, что любые узкие места в ходе процессов их создания могут привести к нежелательным результатам. Так, удлинение сроков разработки чревато удорожанием конечного продукта, не выявленные в ходе тестирования ошибки приводят к снижению правильности его работы и потерям в производительности и эффективности. Наличие ошибок, непонятных сообщений и рекомендаций, непривлекательный интерфейс раздражает пользователей, что в итоге приводит к выбору другого, более качественного продукта конкурента, и компания-разработчик рискует потерять свою долю рынка.

Все это означает, что, качество программного обеспечения (ПО), а также способы его эффективной оценки приобретают важнейшее значение, при этом следует иметь в виду, что один и тот же продукт может иметь несколько различных оценок качества, которые могут использоваться для различных целей и условий применения. В целом качество программных систем необходимо рассматривать как иерархическую совокупность свойств, размещенных на различных уровнях. Изучение взаимосвязи между показателями, входящими в состав комплексной оценки качества является основой теоретического обоснования выбора ее структуры и способов измерения и анализа отдельных атрибутов и метрик при квалиметрических исследованиях отдельных свойств в зависимости от конкретных условий применения.

1 КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Под общим понятием *качества* понимается количественная или качественная степень соответствия присущих объекту характеристик потребностям или ожиданиям пользователей, которые могут быть установлены как рекомендательные или обязательные. Как показывает международная практика, наиболее успешно задача обеспечения качества решается на основе системного подхода, предусматривающего создание на предприятиях систем управления качеством в виде совокупности организационной структуры, методик, процессов и ресурсов.

1.1 Системы управления качеством

Управление качеством включает методы и мероприятия оперативного характера, обеспечивающие мониторинг и устранение причин неудовлетворительной работы объекта в ходе его применения при обеспечении основных критериев экономической эффективности. При этом применяются измерительные процедуры, экспертное оценивание, анализ показателей качества, а также исследование основных технологических процессов, в ходе которых формируются потребительские свойства продукции. В ходе этих процедур производится накопление и обработка соответствующей информации с применением статистических и других математических методов. В случае выявления отклонений от принятых требований выполняются корректирующие действия для устранения причин.

Система управления качеством продукции [2, 3], в том числе и ПО, должна объединять все этапы жизненного цикла, включающих: маркетинговые исследования рынка, разработку технических требований, проектирование и изготовление, материально-техническое обеспечение,, проведение испытаний и тестирования, реализацию, установку и эксплуатацию, техническое обслуживание и утилизацию после использования.

Основными операциями процессов циклического управления качеством продукции различного назначения являются:

1) разработка нормативно-технической документации в виде программ управления, планирования и повышения качества;

2) сбор и аналитическая обработка информации о состоянии исследуемого объекта, определяющего его характеристики качества;

3) формирование управленческих решений по повышению уровня качества и реализация необходимых воздействий на объект;

4) изучение и анализ изменений в состоянии объекта, вызванными принятыми решениям, с целью оценки их правильности и эффективности.

Одним из ключевых элементом в управлении качеством является стандартизация, целью которой является разработка и использование системы нормативно-технической документации, которая определяет требования и методы оценки показателей качества. Данные требования к качеству продукции устанавливаются государственными и международными стандартами, а также техническими условиями, необходимыми для производства и выпуска продукции.

Система обеспечения качества должна удовлетворять следующим требованиям:

- необходимо удовлетворять требованиям потребителя данной продукции или услуги;
- дефекты, ошибки и недостатки в работе объекта следует предупреждать, а не выявлять после их возникновения и обнаружения;
- необходимо предусматривать эффективные способы обнаружения и установления причин выявленных дефектов, ошибок и недостатков;
- следует гарантировать невозможность поступления в дальнейшую эксплуатацию продукции с выявленными отклонениями показателей качества;
- необходимо обеспечивать постоянное совершенствование характеристик качества продукции и самой системы управления качеством.

При формировании различных подходов к системам обеспечения качества продукции произвольного назначения, международное сообщество рекомендует опираться на концепцию всеобщего управления качеством TQM (*Total Quality Management*), которая представляет собой обобщенную систему, основанную на производстве качественных с точки зрения заказчика продукции и услуг [4]. TQM направлена на планомерное достижение стратегической цели организации при помощи непрерывного улучшения эффективности выполняемых работ. В этот процесс следует вовлекать каждого сотрудник организации или предприятия, термин «качество» должен означать наилучшее удовлетворение потребностей клиента, а термин «управление» должен относится к сотрудникам и процессам, используемым для достижения заданного уровня качества. Стратегия TQM проста и универсальна – выяснить нужды потребителя и удовлетворить их.

Основные принципы обеспечения качества, а также требования к системам менеджмента качества организаций и предприятий изложены в серии международных стандартов ИСО 9000, которая была разработана Международной организацией по стандартизации [3, 5]. Стандарты серии ИСО 9000 приняты в большинстве страна мира в качестве национальных стандартов и применяются к любым предприятиям, независимо от их размера, форм собственности и сферы деятельности. В России новые стандарты ИСО 9001:2000 утверждены в качестве Национальных стандартов (ГОСТ) с 15 августа 2001 г.

Поэтому в настоящее время в Российской Федерации действует абсолютно идентичная международным стандартам серии ИСО 9001 версии 2000 года серия стандартов ГОСТ Р ИСО 9000 версии 2001 года.

Показатели качества продукта или процесса, а также способ их оценивания, зависят от цели, потребителя и условий внешней среды, при которых используется результат оценки. При этом один и тот же объект может быть представлен различными оценками качества, применяемыми для различных условиях применения. Отклонения фактических значений показателей качества от требуемых квалифицируются как дефекты или ошибки и представляют собой первичные предпосылки для принятия и реализации решений по улучшению качества.

Стандарт ИСО 9001:2000 уделяет приоритетное внимание процессному подходу к управлению качеством. На рис. 1 представлена общая концепция процессной модели управления качеством. Под процессным подходом в деятельности предприятия по организации и управлению качеством понимается ориентация на существующие бизнес-процессы в рамках действующей организационно-штатной структуры предприятия.

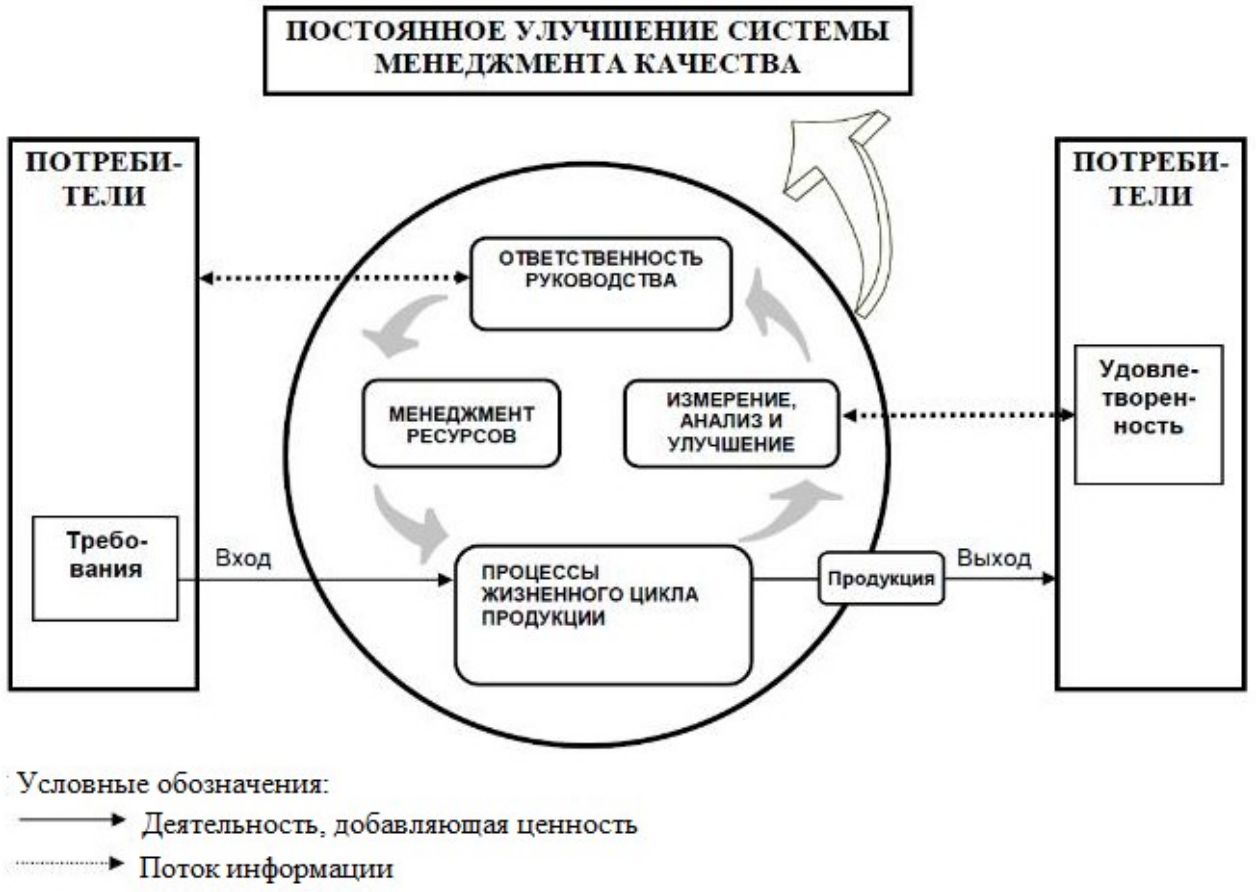

Рис. 1. Процессная модель качества

Процессный подход предназначен для обеспечения взаимных связей всех видов деятельности организации или предприятия, их согласованность и направленность на достижение стратегических целей. Он дает возможность анализировать процессы, совершенствуя и приспосабливая их к изменениям, а также облегчает управление организацией и объединяет людей в командной работе. Схематически такая модель приведена на рис. 2

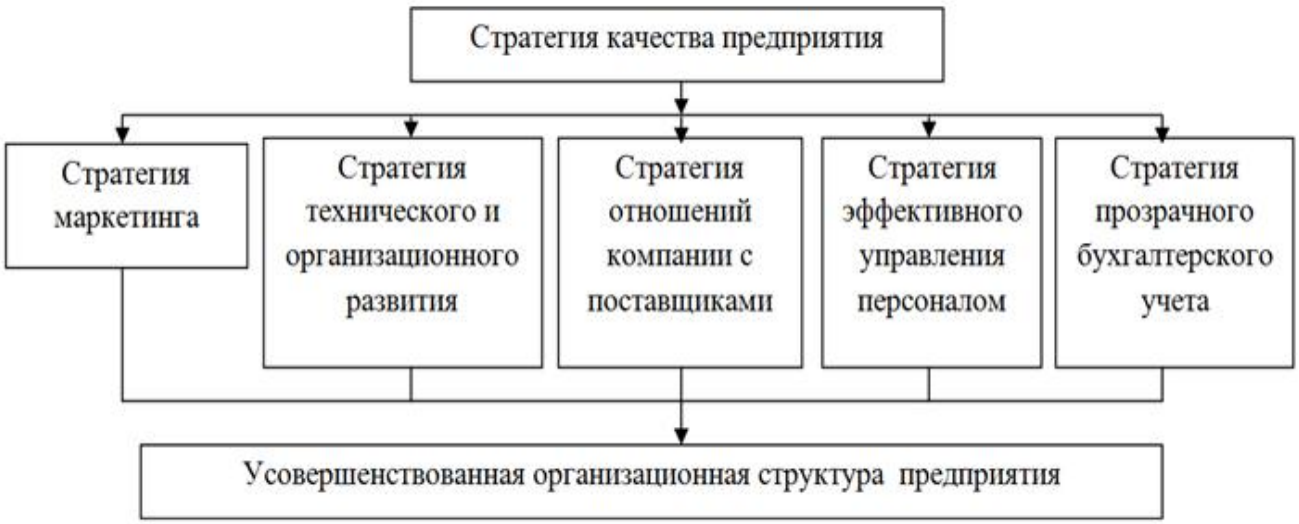

Рис. 2. Стратегия управления качеством

1.2 Стандартизация качества ПО

Важнейшей проблемой обеспечения качества программных систем является применение формальной системы характеристик качества и методологии их оценки [2, 6, 7]. Для соответствия качества использования в реальных условиях проектирования и эксплуатации, наличия технических возможностей компонент программных средств к взаимодействию, способности к совершенствованию и развитию нужно использовать стандарты, определяющие как сами характеристики качества, так и способы их оценки.

Показатели качества ПО установлены в ГОСТ Р ИСО/МЭК 9126-93 «Информационная технология. Оценка программной продукции. Характеристика качества и руководства по их применению» [6]. Российским аналогом этого стандарта является ГОСТ 28195 – 99 «Оценка качества программных средств» [7]. Существование двух действующих стандартов, которые нормирующих одну и ту же системы показателей и способы их оценки, поднимает вопрос об их правильном и гармоничном использовании. Рассмотрим и характеризуем далее каждый из перечисленных стандартов.

ГОСТ 28195-99 определяет общие положения по оценке качества программных систем, номенклатуру и применяемость характеристик качества. В нем отмечается, качество, основы которого формируются при проек-

тировании и производстве программных систем, можно обеспечивать двумя методами:

- при помощи применения регламентированных технологий и систем обеспечения качества процессов проектирования и производства, которые бы предотвращали дефекты и минимизировали ошибки и гарантировали требуемое качество ПО в процессе его создания;
- посредством использования итогового контроля и тестовых системных испытаний готовых программных продуктов с целью исключения из поставки и направления на доработку программных модулей, не соответствующих заданным показателям качества.

Таким образом необходимо управлять качеством программной системы на каждой стадии ее жизненного цикла при условии рассмотрения разнообразных аспектов качества и учета изменений представлений о качестве в ходе процесса проектирования и эксплуатации.

Стандарт ГОСТ Р ИСО/МЭК 9126-93 предлагает рассматривать качество программного продукта в соответствии со по стадиями его жизненного цикла с использованием следующий уровней.

1) *Целевое качество*, под которым понимается необходимое и достаточное качество, отражающее реальные потребности пользователя. Так как требования к программной системе, заявленные заказчиком, не всегда изначально полностью отражают реальные потребности пользователей, использующих этот программный продукт, то эти потребности могут изменяться после того, как они зафиксируются в документации проекта. Целевое качество рассматривается проектировщиками как концептуальное свойство, которое не может и не должно полностью определяться в начале проекта, его следует воспринимать как ориентир. Требования к целевому качеству следует по возможности включать в техническое задание в части спецификации требований к качеству программной системы (ПС), однако оценивать их следует по завершении разработки программного продукта посредством измерения качества эксплуатации.

2) *Затребованное (установленное)* качество продукта представляет уровень характеристик внешнего качества, фактически заявляемый в техническом задании в части описания требований к качеству и предназначен для использования как цель при проверке ПС. При формулировании таких требований ко всем или некоторым характеристикам качества в спецификации наряду с оптимальными значениями характеристик следует указываться и предельные значения (минимальные и/или максимальные). Это будет способствовать заказчикам и проектировщикам избежать превышения стоимости и увеличения сроков разработки.

3) *Качество программного проекта* является внутренним уровнем

качества программной системы, которое касается представлений в описании основных частей или всего проекта в целом таких элементов как архитектура ПО, структуры программного комплекса, модели пользовательского интерфейса, алгоритмов и т.п. Качество проекта лежит в основе качества программной системы, оно, как правило, может быть лишь незначительно улучшено в ходе изменения программного кода и тестирования.

4) *Оцененное (или прогнозируемое) качество* представляет собой уровень, на котором оценивается или предсказывается комплексное качество конечного программного продукта на основании характеристик качества программного проекта.

5) *Качество поставляемого продукта* характеризует степень готовности программного продукта к поставке, которая определяется в результате тестирования в среде моделирования с использование моделируемых данных.

6) *Эксплуатационное качество* является уровнем качества программной системы, которое характеризуется по результатам ее использования. При этом пользователь оценивает только те атрибуты качества, которые видны ему в ходе реального использования, поэтому такое представление о качестве ПО может отличаться от других представлений из-за недостаточного учета особенностей программного окружения и сценариев применения программной системы.

Стандарт ГОСТ Р ИСО/МЭК 9126-93 состоит из 4 частей, в которых излагаются следующие категории:

- модель качества;
- внешние метрики;
- внутренние метрики;
- метрики качества в использовании.

Модель качества классифицирует качество ПО в шестью структурными наборами характеристик – показателями качества ПО. Эти показатели в свою очередь детализируются атрибутами (субхарактеристиками), как показано на рисунке 3.

К основным характеристикам, определяющим модель качества программной системы относятся:

1) *функциональность* – соответствие функциональных возможностей ПО набору назначений, который требуется пользователю;

2) *надёжность* – способность ПО сохранять необходимый уровень качества в определенных условиях в течение заданного периода времени;

3) *практичность* (удобство применения) характеризует объемы работ, необходимых для использования программной системы определенным или предполагаемым кругом пользователей;

4) *эффективность* оценивает соотношение качества функционирования программной системы в сочетании с объемом используемых ресурсов и временной производительностью (скоростью работы);

5) *сопровождаемость* – показатель, определяемый объемом работ, выполняемых для проведения конкретных изменений (модификаций) в процессах сопровождения и продолжающейся разработки;

6) *мобильность* – способность ПО к перенесению из одного аппаратно – программного окружения в другое.

Рис. 3. Показатели качества по ГОСТ Р ИСО/МЭК 9126-93

Таким образом, представление качества ПО является иерархически многоуровневым, где:

- первый уровень отражает комплекс перечисленных выше шести характеристик (показателей) качества ПО;
- второму уровню соответствуют атрибуты для каждого показателя качества, детализирующие его разные аспекты и использующиеся при оценке качества;

- третий уровень определен для измерения и оценки качества с помощью метрик, под которой понимается комбинация способа измерения атрибута и шкалы измерения значений;

- четвертый уровень представляет собой оценочные элементы метрики (веса), используемые для получения количественного значения или качественной оценки конкретного атрибута показателя ПО.

Приведенные характеристики и атрибуты качества ПО используются для систематического описывания требований к нему, определяя, какие конкретные свойства программ хотят обеспечить заинтересованные стороны по заданной характеристике. Этими требованиями определяются следующие аспекты качества:

- что программная система должна делать, например, позволять клиенту оформлять заказы и обеспечить их доставку, обеспечивать контроль качества строительства и контролировать проблемные места;

- насколько ПО должно быть надежным, например, работать 7 дней в неделю и 24 часа в сутки, никакие введенные пользователями данные при отказе не должны теряться;

- степень удобства использования этого ПО, например, пользователь, зная название товара и имея средние навыки работы в Интернет, должен находить нужный ему товар за не более чем 1 минуту;

- насколько программная система должно быть эффективной, например, поддерживать обслуживание до 10000 запросов в секунду, или время отклика на запрос при максимальной загрузке не должно превышать 2 с;

- насколько удобным должно быть его сопровождение, например, добавление в систему нового вида запросов не должно превышать по времени более 3 человеко – дней;

- степень переносимости (мобильности), например, ПО должно работать на операционных системах Linux, Windows 10 и MacOS X, приложение должно взаимодействовать с документами в форматах Microsoft Word и HTML.

Рассуждая о качестве ПО, следует иметь в виду, что взгляды различных участников реализации программного проекта на важность и способы получения тех или иных характеристик качества могут сильно отличаются.

Пользователи как лица, непосредственно взаимодействующие с ПО с целью выполнения определенных задач, оценивают его без исследования его внутренних аспектов того, как это ПО разрабатывалось. Для них большую ценность имеют такие показатели качества ПО как функциональные возможности и практичность. Поэтому при проектировании нужно уделять серьезное внимание обеспечению необходимой функциональности и проектированию пользовательского интерфейса.

Покупатель, который представляет собой физическое лицо или организация, приобретающие ПО и являющиеся владельцем лицензии на его использование, преследует в качестве основной цели как удовлетворение нужд потребителя, к которым относятся оптимизация расходов, связанных с приобретением и внедрением.

Инвесторы, оплачивающие разработку, ожидают, что их вложения принесут доход, который может определяться повышением эффективности работы и производительности труда организации, для которой этот программный продукт предназначается.

Разработчики как лица или организация, непосредственно создающие ПО, заинтересованы в том, чтобы создаваемый ими продукт действительно удовлетворяет заданным требованиям. Для них промежуточные артефакты проекта важны не меньше, чем качество конечного продукта.

Руководитель программного проекта заинтересован в общем оценке качестве больше, чем в оценке некоторой конкретной характеристике качества, он нуждается в определении показателей, отражающих коммерческие требования для индивидуальных характеристик.

1.3 Оценка и анализ качества программных систем

Качество разрабатываемого программного обеспечения оценивают вектором шести изложенных выше комплексных показателей.

Атрибуты качества программной системы, как характеристики качества следующего уровня, детализирующего уровень комплексных показателей, измеряются с использованием метрик качества. Под метрикой понимается сочетание его сущности, конкретного метода измерения как способа получения количественной оценки, и шкалы измерения, при помощи которой получаемые значения структурируется и систематизируются. Метрика использует меру, то есть переменную, которой присваивается значение в результате измерения (см. рис. 4). В свою очередь, если метрике необходима дальнейшая детализация на дополнительные, наиболее мелкие характеристики качества, то используется последний предусмотренный стандартом [7] уровень оценочных элементов

Наиболее подход к моделированию качества ПО состоит в том, что сначала идентифицируется небольшой набор характеристик - показателей качества самого высокого уровня абстракции, который затем в направлении «сверху вниз» разбивается на наборы подчиненных атрибутов и далее на метрики и оценочные элементы.

Оценка качества программной системы производится с использованием шести базовых показателей, каждый из которых детализируется несколькими нормативными атрибутами (субхарактеристиками).

Рис. 4. Система измерения качества

Рекомендуется следующая общая схема процесса оценки показателей качества программных систем:

1) формирование исходных требований для анализа качества ПО, включая определение целей, выделение адекватных реальным системам показателей и требуемых значений атрибутов качества, спецификацию типа метрик;

2) селективный выбор метрик качества с установлением рейтингов и уровней приоритета как показателям и атрибутов, так и метрикам и оценочным элементам, определение критериев для проведения измерений и оценки качества;

3) планирование процесса анализа и проектирование процедур оценки характеристик и атрибутов качества в соответствии с жизненном циклом программной системы;

4) выполнение измерений для метрик и оценочных элементов, сравнение результатов с критериями и требованиями, обобщение и анализ результатов;

5) формирование рекомендаций по повышению качества программной системы.

Для каждой характеристики качества следует сформировать не только меры, но и шкалы измерений с выделением требуемых, допустимых и неу-

довлетворительных значений. Реализация процессов оценки должна коррелировать с этапами жизненного цикла программного средства.

1.4 Метрики

Для определения необходимого и достаточного набора средств измерения качества ПО используются разные подходы. Система оценивания показателей качества включает метрики, оценочные элементы и измерительные модели, применяемые для получения количественных значений характеристик. Следует отметить, что в настоящее время в программной инженерии еще не сформирована окончательно функционально полная система метрик.

В процессе определении требований к ПО определяются соответствующие им внешние характеристики и атрибуты, определяющие разные стороны качества продукта в определенной среде. При этом определяются сами метрики, способы их оценки и диапазоны значений мер для измерения или экспертной оценки отдельных атрибутов качества.

Метрики, в соответствии со стандартом [7], определяются на всех этапах ЖЦ как промежуточные, внутренние и метрики. По сути набор метрик качества представляет собой систему измерений качества конкретной программной системы. Измерения или экспертное оценивание могут проводиться на как на уровне показателей качества ПО, так и на уровне детальных характеристик. В первом случае оценка качества позволяет непосредственно сравнивать программы, при этом измерительные процедуры проводятся на основе субъективных оценок свойств. Во втором случае характеристики можно измерить более объективно и достоверно, однако оценка качества системы в целом будет связана с субъективной интерпретацией получаемых оценок [2, 8].

В время выполнения оценки качества ПО с использованием метрик выделяют два основных процесса:

1) поиск набора метрик, который наиболее хорошо характеризует необходимые качественные свойства программной системы;

2) применение метрик для получения количественных оценки процессов и факторов проектирования ПО, позволяющих проанализировать условия его разработки.

По виду информации, которая формируется при оценке качества ПО метрики можно разделить на следующие группы;

1) метрики, которые оценивают отклонение от нормы характеристик качества проектных материалов и документов, с их помощью можно установить полноту заданных технических характеристик.

2) метрики, применяемые для прогнозирования качества проектируемого ПО, они задаются на множестве возможных вариантов решений поставленной задачи и их ;

3) метрики, предназначенные для принятия решений о соответствии созданного программного продукта заданным требованиям.

Применяются три основных типа метрик:

1) метрики программного продукта, используемые для измерения или экспертной оценки его свойств;

2) метрики процесса, используемые при оценки свойства процесса жизненного цикла ПО;

3) метрики использования.

Кроме того, метрики программного продукта могут быть внешними, характеризующими свойства, видимые пользователю, и внутренними, обозначающие характеристики качества, видимые только проектной команде.

К внешним метрикам программного продукта относятся метрики:

- надежности, которые используются для определения числа дефектов и ошибок;
- функциональности, устанавливающие наличие и правильность реализации требуемых функций программной системы;
- сопровождения, измеряющие такие ресурсы как скорость, память, окружающая среда;
- применимости, позволяющие определить степени доступности ПО для изучения и использования;
- стоимости, используемыми для оценки стоимости.

Внутренние метрики ПО подразделяются на метрики:

- размера, применяемые для измерения внутренних характеристик ПО;
- сложности, используемые для оценки его сложности;
- стиля, которые применяются для оценки подходов и технологий создания отдельных компонентов ПО и его документов.

Внутренние метрики, которые могут определить производительность программной системы, являются релевантными по отношению к внешним метрикам.

Как внешние, так и внутренние метрики формируются на этапе определения требований к ПО, они являются объектами планирования и управления при достижении заданного качества программного продукта.

Метрики программного продукта могут описываться набором моделей для установки различных характеристик, их значений или прогнозирования. Для проведения измерений, как правило, необходима калибровка метрик на ранних этапах проекта. Общей мерой при этом является степень трассируемости, определяемая числом трасс, создаваемых с помощью моделей сценариев

и оценкой количества требований, сценариев и действующих лиц, объектов, включенных в сценарий, параметров и операций объекта и др.

Стандарт ГОСТ Р ИСО/МЭК 9126-93-2 определяет следующие типы мер:

- меры размера в разных единицах измерения, такие как число функций, строк в программе, размер дисковой памяти и др.;
- меры времени, затрачиваемого на реализацию функций системы, выполнения компонента и др.;
- меры усилий, такие как производительность труда, трудоемкость и др.;
- меры учета, оценивающие количество ошибок, число отказов, ответов системы и др..

Особой мерой является уровень использования повторных компонентов, который измеряется как отношение совокупного размера готовых компонентов программного продукта к размеру системы в целом. Эта мера может применяться и при оценке стоимости и качества ПО. Примерами таких метрик являются:

- общее число и число повторно используемых объектов;
- общее число, число повторно используемых и новых операций;
- количество классов, наследующих объектные операции;
- количество классов, от которых зависит данный класс;
- число пользователей класса или операций и др.

При оценке количественных показателей некоторых характеристик часто используются среднестатистические метрики, например, среднее число операций в классе, наследников или операций класса и др.. Такие меры, как правило, в значительной степени являются субъективными и определяются опытом и знаниями экспертов, которые занимаются получением этих количественных оценок.

Классическим примером используемых внешних метрик являются метрики Холстеда [9], которые представляют собой характеристики ПО, выявляемые на основе статической структуры программы на конкретном языке программирования, таких как число вхождений наиболее часто встречающихся операндов и операторов, длина описания программы как сумма числа вхождений всех операндов и операторов и др.

Эти характеристики можно использовать для вычисления времени программирования, уровень структурированности и качества программы, оценки языка программирования с точки зрения использования абстракций, ориентации на проблему и так далее. Для практического использования можно рекомендовать такие метрики процесса программирования как:

- общее время разработки и отдельно время для каждой стадии;
- время выполнения работ в заданном процессе;
- количество найденных ошибок при инспектировании;

- стоимость оценки качества;
- время модификации моделей;
- стоимость процесса программирования.

Для измерения степени удовлетворения потребностей пользователя при решении его задач применяются метрики использования. Они позволяют оценить не конкретных свойства программного продукта, а эксплуатационное качество как результаты его эксплуатации. В качестве примеров таких метрик можно привести следующие: точность и полнота реализации задач пользователя, затраченные ресурсы (трудозатраты, производительность и др.) на эффективное решение пользовательских задач. Проверка же требований пользователя должна производиться с использованием внешних метрик.

В современной мировой практике применяется несколько сотен метрик ПО. Эту значительный набор можно систематизировать по следующим шести направлениям:

1) оценка топологической и информационной сложности ПС;
2) оценка надежности ПС с возможностью прогнозирования ситуации отказов;
3) оценка производительности ПО;
4) оценка уровня языковых и инструментальных средств проектирования ПО;
5) оценка восприятия и понимания программной документации;
6) оценка производительности труда разработчиков при планировании сроков разработки программ и распределения работ.

Для получения количественной оценки метрики нуждаются в использовании измерительные шкал. Основными типами таких шкал являются номинальные, порядковые, интервальные, а также относительные и абсолютные.

Номинальные шкалы используются метриками, которые классифицируют ПО на типы по признаку наличия или отсутствия некоторой характеристики без использования градаций. Порядковые шкалы применяются метриками, которые могут ранжировать характеристики качества путем сравнения с опорными значениями. Интервальная шкала свойственна метрикам, которые показывают не только относительные значения характеристик, но и то, насколько далеко они расположены друг от друга.

Абсолютная шкала указывает на фактическое значение величины (например, число ошибок в программе равно 10). Относительная шкала применяется в случае когда метрика оценивается отношением фактического значения величины к некоторому опорному значению или значению, с которым производится сравнение.

1.5 Контрольные вопросы и задания

1) Что понимают под моделями качества процессов разработки программного обеспечения? Для чего они разработаны?

2) Что определяет показатель качества? Как он формируется?

3) Что понимают под моделями качества процессов разработки программного обеспечения? Для чего они разработаны?

4) Какие виды метрических шкал применяются для измерения метрик качества ПО?

5) В чем разница оценки качества аппаратного и программного обеспечения?

6) Дайте определение эффективности программного средства.

7) Каким образом связаны между собой технология программирования и качество ПО?

8) Каковы критерии применения различных типов метрик (программного продукта, процесса или использования) в оценке качестве ПО?

2 НАДЕЖНОСТЬ ПРОГРАММНО – АППАРАТНЫХ КОМПЛЕКСОВ

Надежность относится к наиболее универсальным и системным показателям качества объектов самого широкого назначения, определяемым на всех стадиях жизненного цикла. Системность надежности заключается в том, что для сложного изделия она определяется не только надежностью отдельных компонентов и подсистем, но и их взаимным влиянием. Поэтому надежность программного обеспечения необходимо рассматривать исходя из того, что оно является одной из обеспечивающих подсистем автоматизированных систем обработки информации и управления, наряду с другими подсистемами, такими как, техническое, информационное, математическое и других [10,11].

Свойство надежности этих обеспечивающих подсистем определяется их различной природой, спецификой проектирования, изготовления и использования. Так надежность электро-механических или электронных технических средств определяется не только дефектами, вызванными ошибками в проектировании и технологическими недоработками, но и дефектами, появляющимися в процессе эксплуатации в результате износа, старения, влияния внешней среды и неправильных действий персонала. Все это приводит к тому, что с течением времени надежность технических средств в целом снижается. Программное же обеспечение не чувствительно к эксплуатационным факторам, поэтому его надежность определяется в основном устранимыми ошибками системного проектирования и программирования, что обеспечивает повышение показателей надежности на протяжении длительной эксплуатации. Свою специфику с точки зрения надежности имеют и информационное, и математическое и другие виды обеспечивающих подсистем.

2.1 Показатели надежности технических средств

Надежность любого технического устройства, в том числе и электронных компонентов автоматизированных систем обработки информации и управления, является комплексным свойством включающим в себя такие показатели, как безотказность, ремонтопригодность и сохраняемость.

Под *безотказностью* понимается свойство всей системы или отдельного ее элемента сохранять непрерывно работоспособное состояние в течение некоторого времени или наработки, под которой понимают объем работы системы.

Сохраняемость представляет собой свойство устройства непрерывно находиться в исправном и работоспособном состоянии во время хранения.

Ремонтопригодность характеризует приспособленность системы к предупреждению, обнаружению и устранению причин возникновения отказов

за счет выполнения технического обслуживания и ремонта. Таким образом, надежность является фундаментальным понятием, в сферу рассмотрения которого входят все стороны технической эксплуатации элементов и систем.

Поскольку отказы и сбои как элементов, так и систем являются случайными событиями, то основным математическим аппаратом, используемым при исследовании надежности, является теория вероятностей и математическая статистика. Таким образом, показатели надежности рассматриваются вероятностные величины.

К наиболее распространенным количественным характеристикам надежности систем различного назначения относятся [10,11]:

- вероятность безотказной работы в течение определенного времени – $P(t)$;
- средняя наработка до первого отказа – $T(cp)$;
- вероятность отказа – $Q(t)$;
- наработка на отказ – $t(cp)$;
- частота отказов – $a(t)$;
- интенсивность отказов – $\lambda(t)$ и другие.

Вероятность безотказной работы можно рассчитать по статистическим данным об отказах с использованием следующего выражения:

$$\hat{P}(t) = (N_0 - n(t))/N_0, \qquad (1)$$

где N_0 – количество объектов в начале испытания; $n(t)$ – число объектов, отказавших за время t; $\hat{P}(t)$ – статистическая оценка $P(t)$.

Часто более удобной на практике характеристикой является вероятность отказа $Q(t)$, то есть вероятность того, что время до случайного проявления отказа меньше заданного времени t, которая является дополнением вероятности безотказной работы до единицы:

$$Q(t) = 1 - P(t) \qquad (2)$$

Эта функция $Q(t)$ соответствует интегральной функции распределения времени наступления отказа $F(t)$:

$$Q(t) = F(t) = \int_0^t f_t(x)dx, \qquad (3)$$

где $f_t(x)$ – функция плотности распределения времени, прошедшего до наступления отказа; x – переменная интегрирования.

Тогда справедливы выражения:

$$Q(t) = 1 - P(t) = 1 - \int_0^t f_t(x)dx = \int_t^\infty f_t(x)dx \qquad (4)$$

Частота отказов представляет собой производную от вероятности безотказной работы, ее называют плотностью распределения времени безотказной работы:

$$a(t) = Q'(t) = -P'(t) \qquad (5)$$

Частота отказов связана с ВБР и вероятностью появления отказа следующие зависимостями:

$$Q(t) = \int_0^t a(t)dt \qquad (6)$$

$$P(t) = 1 - \int_0^t a(t)dt \qquad (7)$$

Интенсивность отказов $\lambda(t)$ показывает, насколько часто возникают отказы. Оценку этой характеристики определяют выражением:

$$\lambda(t) = a(t)/P(t) \qquad (8)$$

Связь между интенсивностью отказов и ВБР определяется зависимостью:

$$P(t) = e^{-\int_0^t \lambda(t)dt} \qquad (9)$$

Если $\lambda(t) = \lambda = \text{const}$, то $P(t) = e^{-\lambda \cdot t}$ и $a(t) = \lambda \cdot e^{-\lambda \cdot t}$ соотношение характеризует экспоненциальное распределение безотказной работы.

Время, прошедшее между соседними отказами также является непрерывной случайной величиной, которая характеризуется некоторым законом распределения. Зависимость показателей надежности от времени описывается с помощью некоторой математической модели надежности позволяющей рассчитывать показатели надежности и характерной для различных типов систем и элементов. Простейшие модели имеют вид математических формул, определяющих распределения вероятностей. Так, при анализе надежности технических систем применяются такие модели законов распределения как экспоненциальный, нормальный, Рэлея, Пуассона, Вейбулла и др. [11].

Самой распространенной вероятностной моделью надежности является экспоненциальная модель распределения времени до отказа, которая выражается зависимостью:

$$P_э(t) = e^{-\lambda \cdot t}, \qquad (10)$$

где λ – параметр модели.

Для этого закона распределения средняя наработка на отказ равна:

$$T_{ср.} = \int_0^\infty e^{-\lambda t} \cdot dt = \frac{1}{\lambda}, \qquad (11)$$

а частота отказа выражается формулой:

$$a_e(t) = -dP(t)/dt = \lambda e^{-\lambda t} \qquad (12)$$

Функция интенсивности отказов при экспоненциальной модели равна:

$$\lambda_e(t) = a_e(t)/P(t) = \lambda = \text{const} \qquad (13)$$

2.2 Расчеты надежности технической системы

Расчеты показателей надежности технического устройства, как системы, состоящей из совокупности материальных элементов, проводятся исходя из постулата, что вся система и любой ее элемент могут находиться только в од-

ном из двух возможных состояний – работоспособном и неработоспособном, при этом отказы элементов являются независимыми друг от друга. Работоспособность состояния системы определяется работоспособностью состоянием элементов и их взаимными связями. Расчет надежности любой технической системы в простейшем варианте реализовать как полный перебор всех возможных комбинаций состояний элементов с определением вероятности каждого из них и установлением работоспособных состояний системы.

Такой метод является универсальным и может использоваться при расчете практически любых ТС. Однако при большом количестве элементов системы такой метод является нецелесообразным, а часто и просто нереальным, вследствие огромного объема вычислений, поэтому в практических расчетах применяют более эффективные и экономичные методы расчета, не связанные с существенными объемами вычислений. Эти методы основаны на исследовании структуры ТС и построении эквивалентных структурных схем.

Методика расчета надежности устройств программно-аппаратного комплекса вычислительной системы состоит из следующего набора операций:

- определение типа элемента и его характеристик, формулирование четкого формального понятия отказа для всей системы и его отдельных составных частей;
- анализ структуры устройства, определение основных и вспомогательных блоков системы, составление эквивалентной схема расчета надежности, в которой элементами расчета являются конструктивно оформленные блоки;
- выбор метода расчета с подбором справочных данных и форм представления информации, определение параметров нагрузки элементов и влияния внешней среды;
- определение интенсивности отказов и вероятности безотказной работы каждого элемента;
- определения интенсивности отказов и вероятности безотказной работы системы, расчет других показателей надежности, представление результатов расчета.

Построение эквивалентной структурной схемы при расчете надежности используется слияние нескольких элементов, соединенных определенным образов, в один – эквивалентный, безотказность работы которого в точности будет равна заменяемой совокупности элементов. При этом рассматриваются типовые схемы соединения элементов – последовательное, параллельное и мостиковое.

В системе из *последовательно соединенных элементов* отказ любого элемента приводит к отказу всей системы (рис. 5, а). Так как отказы элементов являются независимыми, то вероятность одновременной безотказной ра-

боты последовательно соединенных n элементов определяется в соответствии с теоремой умножения вероятностей так - вероятность совместного появления независимых событий равна произведению вероятностей этих событий:

$$P(t) = p_1(t)p_2(t)...p_n(t) = \prod_{i=1}^{n} p_i(t) = \prod_{i=1}^{n} (1 - q_i(t)) \qquad (14)$$

Соответственно, вероятность отказа такой системы составит:

$$Q = 1 - P = 1 - \prod_{i=1}^{n} p_i = 1 - \prod_{i=1}^{n} (1 - q_i). \qquad (15)$$

Набор из *параллельным соединением элементов* представляет собой систему, отказ которой произойдет только в случае, когда откажут всех ее элементы (рис.5, б). Такие структуры характерны для устройств, в которых элементы дублируются или резервируются, потому что параллельное соединение применяется как основной метод повышения надежности.

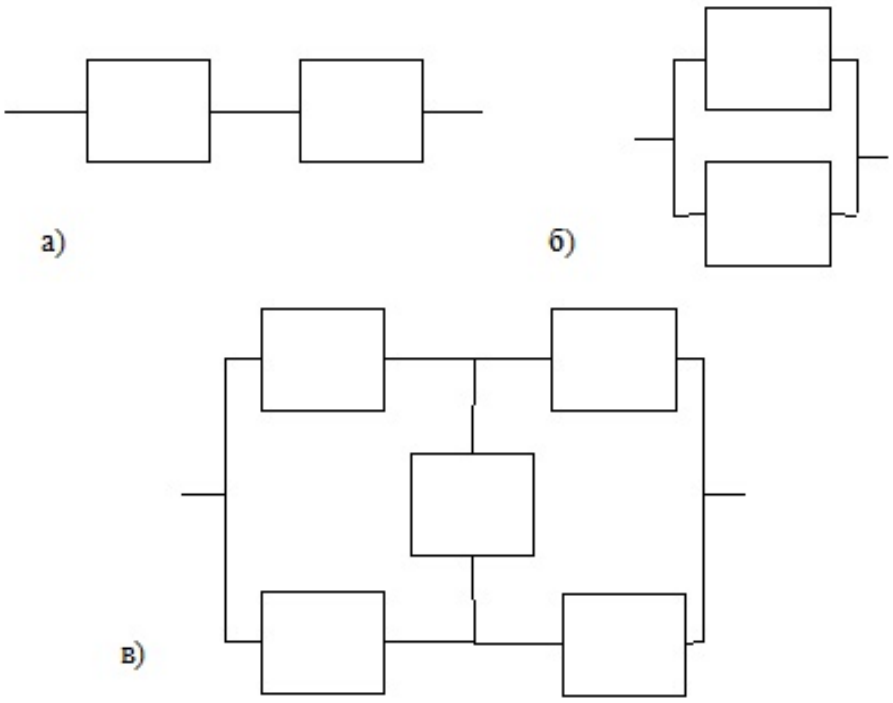

Рис. 5. Системы с последовательным (а), параллельным (б) и мостиковым (в) соединением элементов

При допущении независимости отказов вероятность отказа такой системы может быть найдена так же по теореме умножения вероятностей, но как произведение вероятностей отказа элементов:

$$Q = q_1q_2...q_n = \prod_{i=1}^{n} q_i = \prod_{i=1}^{n} (1 - p_i). \qquad (13)$$

Тогда вероятность безотказной работы параллельно соединенных элементов составит:

$$P = 1 - Q = 1 - \prod_{i=1}^{n} q_i = 1 - \prod_{i=1}^{n} (1 - p_i). \qquad (14)$$

Мостиковая структура (рис. 5, в) не приводится ни к параллельному ни к последовательному соединениям. Она представляет собой параллельно соединенные последовательные цепочки элементов с диагональными элементами, которые включены между узлами параллельных ветвей (рис. 5, в).

Работоспособность мостиковой определяется не только количеством отказавших элементов, но и их положением в структурной схеме. Для расчета надежности мостиковых систем можно воспользоваться методом прямого перебора или *метод логических схем* с применением алгебры логики. Последний сводится к составлению логического выражения, которое является условием работоспособности системы. Для каждого элемента и системы также используются два противоположных логических состояния - отказ или состояние работоспособности.

Реальные технические системы в большинстве своем имеют сложную комбинированную структуру, где отдельные группы элементов могут образовывать последовательные, параллельное, или мостиковые схемы соединений. В этих случаях следует предварительно произвести декомпозицию исходной схемы надежности, разбив ее на простые группы элементов, методика расчета надежности которых известна, после чего заменить эти группы в новой структурной схеме эквивалентными квазиэлементами, вероятности безотказной работы которых будут равны вычисленным вероятностям безотказной работы этих групп. Данную процедуру при необходимости можно выполнять несколько раз, до тех пор, пока получившаяся схема надежности не превратится в структуру, методика расчета надежности является известной и наиболее простой [11].

2.3 Особенности программного обеспечения как объекта надежности

Как уже отмечалось выше, оценка надежности программного обеспечения существенно отличается от приведенных в разделах 2.1 и 2.2 методов расчета надежности технических систем потому, что отказы ПО имеют иную природу и не зависят от условий эксплуатации, внешнего окружения, физического старения и износа элементов и иных причин, характерных для материальных объектов.

Программа представляет собой закодированную в виде инструкций на некотором языке программирования информацию о способе преобразования входных данных в выходные. Ее отказ есть событие, состоящее в прекращении выполнения требуемых функций с учетом заданных ограничений [12].

Основной источник ненадежности программы – ошибки, сделанные разработчиками программ, на разных стадиях проектирования. Истинные причины возникновения ошибок в программных системах состоят в том, что:

- сложность ПО как по объему элементов, так и по структуре и взаимной связности как правило существенно выше, в то время как аппаратная система проектируется из сравнительно небольшого числа типовых элементов;
- используются более значительные наборы входных данных;
- имеется большее число внутренних состояний;
- длиннее зависимости между состояниями во времени;
- аппаратные решения меньше подвержены изменениям;
- для технических систем характерно меньшее взаимное влияние элементов друг на друга;
- программные ошибки в большей степени определяются «человеческим фактором», а аппаратные внешними условиями.

Следует также отметить, что:

- отказы не имеют случайной природы, а возникают тогда, когда повторяются вызвавшие их условия;
- отказы не зависят от времени, ПО, которое не эксплуатируется, не может отказать;
- при оценке надежности ПО не существует понятия «выборка», ибо каждый экземпляр программного модуля или компонента является клоном, точно копирующим дефекты всех других экземпляров.

При оценке и анализе надежности технической системы модели надежности ее элементов, как правило, известны, так как они определяются на этапе их разработки и изготовления. При этом определение надежности элементов производится путем проверки достаточно большого числа этих элементов, с фиксацией сбоев и отказов. Такие модели надежности элементов позволяют получить оценку надежности всего устройства.

При оценке надежности программ:

- может быть использовано большое количество разных моделей надежности, описывающих один и тот же процесс;
- отсутствует универсальная модели надежности, которую можно было бы применить для описания любых программных систем с удовлетворительной точностью;
- в существующих моделях надежности отсутствует возможность учета «вторичных дефектов».

Процедура оценки надежности программной системы усложняется целым набором факторов, к числу которых можно отнести:

- трудность проведения декомпозиции (разделения на части), что приводит к трудностям при измерении надежности каждой части;

- параллельное тестирование двух экземпляров ПО дает почти полностью зависимые результаты;

- параллельное тестирование нескольких экземпляров имеет смысл только при различном сочетании входных данных и управляющих воздействий, однако при этом сложнее оценивать корреляцию проводимых тестов и обрабатывать полученные результаты;

- количество всех возможных состояний для любой нетривиальной программы насколько велико что не позволяет осуществить полный объем тестовых испытаний.

Исходя из приведенных выше аргументов можно заключить, что:

- отказ ПО не может рассматриваться с точки зрения общепринятой теории надежности технических систем, это самостоятельное понятие, применимое к информационным продуктам и связанное с ошибками программирования;

- использование понятия интенсивности отказов также не имеет смысла;

- надежность программы следует рассматривать в контексте информационной достоверности ее работы;

- теория вероятностей не представляет собой метод, который может эффективно использоваться для исследования процессов, происходящие в ПО.

2.4 Ошибки ПО

Дефект или ошибка в программе представляет собой неправильность, погрешность или неумышленное искажение объекта или процесса, что может быть причиной ущерба – неверного результата вычисления, невозможность выполнения требуемой функции, отказ в дальнейшем применении программного продукта и другие последствия.

Для того, чтобы обнаружить факт возникновения ошибки, должно быть известно или задано правильное, эталонное состояние информационного объекта или процесса, относительно которого определяется отклонение. В качестве такого эталона выступает обычно техническое задание со спецификацией требований заказчика. В этом документе должны быть определены состав, содержание и значения результатов использования программы, которые при определенных условиях и исходных данных должен получать пользователь.

Следует отметить, что для программного кода характерной особенностью является отсутствие полностью определенной программы-эталона, которой должны соответствовать результаты работы проектируемой программы. то является существенной особенностью процесса выявления ошибок в ПО. По этой причине установить наличие и локализовать ошибку в программе

путем ее сравнения непосредственно с эталонной невозможно. Кроме того, при отладке и тестировании могут сначала обнаруживаются вторичные ошибки, которые являются последствиями и результаты проявления других внутренних дефектов или недостаточно корректного определения требований и порядка функционирования программы.

Значительное число ошибок вносится на стадии кодирования – эти ошибки находятся в исходном коде программы. Будем рассматривать в основном такие ошибки.

Функциональные ошибки – нарушения программной спецификации (несоответствие функциональным или нефункциональным требованиям). Приводят к ухудшению функциональности ПО (пригодность, точность).

Нефункциональные ошибки – нарушения правил языка программирования, неправильное использование библиотечных функций и т.п. Приводят к снижению надежности (зрелости) и ухудшению функциональности (защищенности):

- ошибки в последовательных программах;
- ошибки синхронизации.

Основные виды нефункциональных ошибок в последовательных программах на языке С:

- ошибки использования неинициализированного, освобожденного указателя или указателя на NULL;
- утечки ресурсов, в том числе динамической памяти;
- ошибочно игнорируемые участки кода;
- ошибки отсутствия инициализации интервальных переменных;
- ошибки выхода за границы статических и динамических объектов;
- отсутствие проверки возвращаемого значения функций.

Сама по себе ошибки в программе является ненаблюдаемой, увидеть и оценить можно не саму ошибку, а результат ее проявления, который по аналогии с техническими системами принято называть отказом. Надежность связана с частотой проявления ошибок, но не с их количеством – разные ошибки имеют разную частоту проявления. При этом, отказ может быть следствием не одной, а сразу нескольких ошибок, причем ошибки могут компенсировать друг друга, при этом может возникнуть ситуация, когда после исправления ошибки интенсивность отказов возрастает. Исправление ошибки, так же как и любое другое изменение ПО приводит к новой программе, характеризующейся уже другими показателями надежности.

Для ПО систем управления объектов, потеря работоспособности которых может повлечь за собой катастрофические последствия, могут использоваться категории тяжести ошибок приведены в табл. 1.

Таблица 1 – Категории тяжести ошибок ПО

Номер категории ошибки	**Наименование категории ошибки**	**Описание последствий проявления ошибки**
IV	Катастрофическая	Проявление ошибки с высокой вероятностью влечет за собой прекращение функционирования ПО (отказ) и может вызвать повреждение системы автоматизации, объекта управления и окружающей среды, а также гибель и травмы людей
III	Критическая	Проявление ошибки с высокой вероятностью влечет за собой прекращение функционирования ПО (отказ) и может вызвать повреждение системы автоматизации, но не угрожает объекту управления, окружающей среде, жизни и здоровью человека
II	Существенная	Проявление ошибки влечет за собой снижение эффективности функционирования ПО и может вызвать прекращение его работы (отказ) без заметного повреждения системы, объекта автоматизации, окружающей среды, жизни и здоровью людей
I	Несущественная	Проявление ошибки может повлечь за собой снижение эффективности ПО, но практически не приводит к возникновению отказа в нем.

2.5 Принципы проектирования надежного ПО

Рассматривая общие принципы обеспечения надежности ПС как один из основных мотивов разработки ПС, определяющим специфику для всем технологических процессов разработки ПС, следует иметь в виду что все они направлены на сокращение количества и степени влияния программных ошибок. При этом используется широкий спектр разнообразных подходов [12, 13], в наиболее распространенным из которых являются следующие:

- предупреждение ошибок, включая их обнаружение и исправление;
- устойчивость к ошибкам;
- проверка на надежность в процессах проектирования;
- снижение сложности программной системы;
- надежные вычисления;
- обеспечение надежной передачи данных;
- обеспечение надежного хранения данных;
- моделирование надежности.

Основной задачей *предупреждения ошибок* является предотвращение появления ошибок в готовых программных продуктах, что является оптимальным путем к достижению надежности ПО. При этом необходимо применять:

1) методы, позволяющие справиться со сложностью программного кода;

2) методы достижения большей точности при обработке числовой информации;

3) методы совершенствования обмена информацией;

4) методы своевременного обнаружения и устранения ошибок;

5) методы обеспечения контроля принимаемых проектных решений.

Такой подход связан с технологией программирования, а именно с. организацией процессов разработки ПС. При этом надо понимать, что гарантировать отсутствие ошибок в ПС невозможно, поэтому в рамках этого подхода надо стремиться к достижению приемлемого уровня надежности ПС.

Важным является использование методов и средств самостоятельного обнаружения ошибки или отказа в процессе выполнения программы. Немедленное обнаружение имеет такие преимущества как возможность минимизации влияния ошибки и устранение затруднений для программиста, которому придется получать информацию об этой ошибке, локализовать ее и исправлять.

Обнаружение ошибок должно осуществляться по признакам, к которым относятся следующие:

- потеря функциональных возможностей;
- авария системы: зацикливание, останов без какой-либо индикации, ошибки, требующие перезагрузки и т. д.;
- программные ошибки пользователя;
- уничтожение данных;
- аварийный отказ модулей программной системы;
- нарушение защиты данных;
- снижение производительности.

Ошибка в программе может быть не только обнаружена в процессе ее выполнения, но и исправлена с ликвидацией последствий наступившего отказа, что предполагает необходимость использования в программной системе соответствующие *средства самоисправления*.

Обеспечение *устойчивости программы к ошибкам* означает, что программная система должна содержать средства, которые могут локализовать область и степень влияния отказа программы, сократив тем самым влияние неприятных последствий, а также исключить катастрофические последствия отказа. Применение такого подхода часто является весьма затруднительным, потому что многие простые методы обеспечения устойчивости, используемые в технических системах неприменимы в программировании. Например, способ дублирования (резервирования) отдельных блоков и устройств в программной системе невозможен, потому что выполнение двух копий одной и той же программы все равно будет приводить к одинаковому эффекту - правильному или неправильному. Кроме того, добавление в ПО до-

полнительных средств приводит к ее существенному усложнению, что в какой-то мере мешает методам предупреждения ошибок.

В целом методы обеспечения устойчивости программ к ошибкам можно разбить на три группы:

- методы *динамическая избыточность*, например голосования, когда данные обрабатываются независимо несколькими идентичными устройствами, и полученные результаты сравниваются с целью установления одинаковости результата;
- методы *отступления* или сокращенного обслуживания, применяемые обычно лишь тогда, когда для программной системы существенно важно корректно закончить работу, что можно реализовать путем загрузки и выполнения специальный фрагмент программы, призванного в порядке подстраховки систему обеспечить завершение всех управляемых системой процессов без аварийных ситуаций;
- методы *изоляции ошибок* с целью не позволить последствиям ошибки выйти за пределы как можно меньшей части системы ПО.

Из этих методов для большинства программных систем применим только последний, изоляция ошибок. Основные правила изоляции ошибок:

1) не должно быть возможности непосредственно ссылаться в одной программы или ее части на другую программу (часть программы), вносить изменения в данные другой программы;

2) программы и данные, которые они используют, должны быть защищены от операционной системы так, чтобы ошибки в самой операционной системе не приводили бы к случайному изменению кода прикладных программ или их данных;

3) программы не должны иметь возможность остановить систему.

Среди методов *борьбы со сложностью* программных систем известны два наиболее общих метода:

- обеспечение независимости компонент системы;
- использование в системах иерархических структур.

Независимость компонент можно обеспечить путем разбиения программной системы на такие части, между которыми должны быть как можно меньше связей. Основным технологическим приемом для реализации этого метода является модульное программирование. Минимизировать и локализовать связи между отдельными компонентами можно также путем использования иерархических структур, при этом связи будут допускаться только между компонентами, принадлежащими смежным уровням иерархии. Это делает целесообразной декомпозицию системы на подсистемы и далее на программные модули и независимые процедуры .

Совершенствование технологии программирования достигается целым рядом вспомогательных средств, например, использование формального описания программных моделей при помощи языка UML, что является одним из эффективных способов повышения надежности программных систем. Процессы проектирования ПО на всех своих стадиях должны включать процедуры, способствующие выявлению ошибок и оценки надежности программной системы. Общая схема технологической модели проверки программ на надежность представлена на рис. 6.

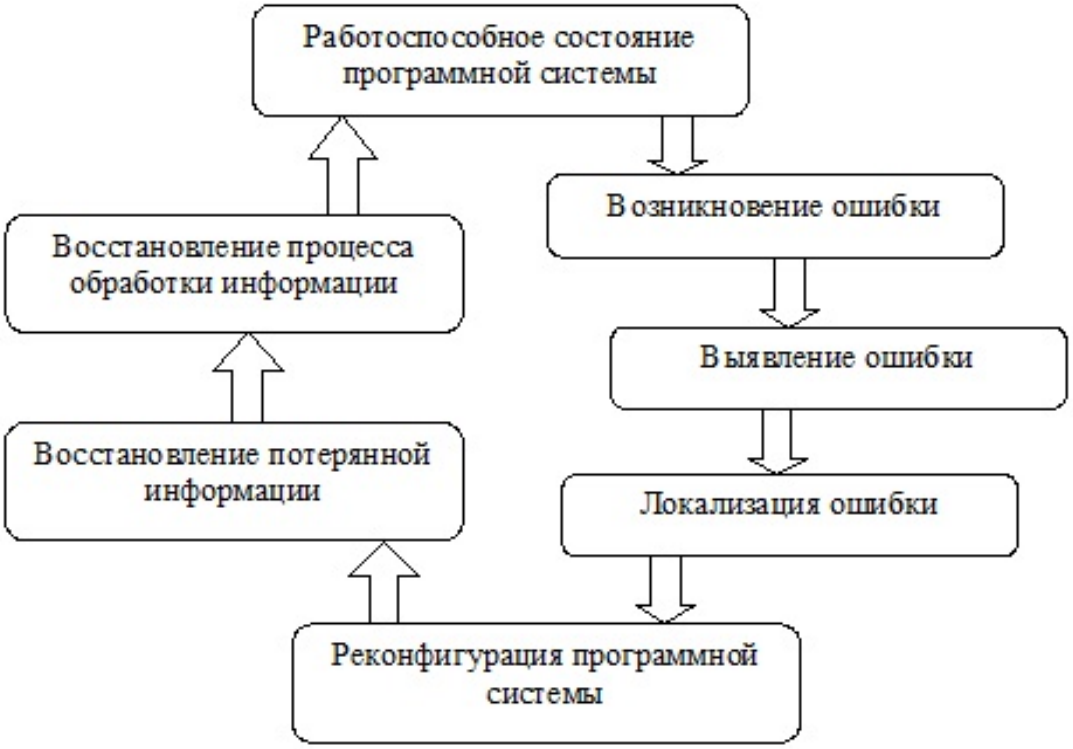

Рисунок 6 – Процесс проверки программной системы на надежность

Помимо обеспечения достаточной точности вычислений при точно заданных исходных данных всегда существовала проблема *надежности результата вычислений*, в случае когда точность коэффициентов математической модели является ограниченной. Например, если в инженерных расчетах исходные данные известны с точностью до третьей значащей цифры, то означает ли это, что после вычисления есть уверенность в достоверности третьей, второй, а иногда даже и первой значащей цифры?

Результат вычисления в математической модели будет надежен только тогда, когда решение будет зависеть от параметров математической модели непрерывно. Когда такой непрерывной зависимости нет, то даже самые малые, неизбежные на практике, погрешности в определении коэффициентов могут приводить к существенным отличиям ожидаемого поведения информационного объекта от реального. В этих случаях результаты расчета не надеж-

ны. Потому одним из способов повышения надежности вычислений является применение алгоритмов, не чувствительных к нарушениям вычислительного процесса, например использование метода алгоритмической избыточности.

Проблема *надежной передачи* данных является одной из центральных для компьютерных сетей и проявляется не только на транспортном, но также на сетевом и прикладном уровнях. Решение этих проблем относится к одной из ключевых задач сетевых технологий и лежит в основе реализации сетевых протоколов и служб [13]. Аналогично *надежность хранения* данных есть одна из ключевых задач систем хранения данных и обеспечивается методами обеспечения надежности как на физическом уровне (контрольное считывание, дублирование дисков), так и на программном уровне СУБД (разграничение прав доступа, контроль завершенности транзакций и др.).

2.6 Модели надежности программных систем

Основным математическим инструментом, который используется для определения и прогнозирования показателей надежности, является модель надежности программной системы, которая строится для установления зависимости надежности от заранее известных или вычисленных в ходе создания программного средства параметров. При этом определение показателей надежности следует рассматривать в единстве процессов предсказания, измерения и оценивания.

Предсказание представляет собой процесс определения показателей надежности проектируемого программного средства. *Измерение* - это процесс определения количественных показателей надежности на основе анализа информации о временных промежутках между отказами, которые происходят при выполнении программ в условиях тестирования. *Оценивание* также представляет собой процедуру определения количественных показателей надежности, но полученными при анализе работы программной системы в реальных условиях эксплуатации.

Для решения этих задач можно использовать аналитические или эмпирические модели надежности программных систем. Если *аналитические модели* позволяют рассчитать количественные показатели надежности на основе информации о поведении системы во время тестирования, то *эмпирические модели* базируются на знании структурных и поведенческих особенностей ПО. Они описывают зависимость показателей надежности от количества межмодульных связей, числа циклов в модулях, отношения длины последовательных участков кода к количеству точек ветвления и так далее. При этом эмпирические модели часто не дают результирующих значений показателей.

Аналитическое моделирование надежности программной системы состоит из четырех основных этапов:

1) определение стратегии и конкретного плана процесса тестирования программного средства исходя из задачи обнаружения ошибок;

2) выбор или создание математической модели, основанной на принятой стратегии и плане тестирования;

3) выбор параметров моделей на основе полученных данных;

4) расчет количественных показателей надежности с использованием построенной модели.

Аналитические модели могут относиться к двум типам: динамическим или статическим. Если в *динамических моделях* надежности ПО поведение программы как процесс появления отказов рассматривается во времени, то в *статических моделях* появление отказов не связывают со временем, а учитывается только зависимость количества ошибок от числа тестовых прогонов или зависимость количества ошибок от состояния входных данных.

Принципиальное отличие статических моделей от динамических заключается в том, что время появления ошибок в процессе тестирования в них не учитывается и отсутствуют предположения о поведении функции риска в процессе тестирования. Статические модели строятся на методах математической статистики.

Рассмотрим далее кратко наиболее распространенные модели надежности программных систем [13, 14].

Модель Джелинского-Моранды. Эта аналитическая модель, иначе называемая моделью роста надежности, относится к динамическим моделям непрерывного времени и основана на следующих предположениях:

- время, прошедшее от предыдущего до следующего отказа распределено экспоненциально;
- все ошибки в программе равновероятны, не зависят друг от друга и имеют одинаковую степень важности;
- исправление ошибок не привносит в программу новых ошибок;
- интенсивность отказов пропорциональна числу еще оставшихся в программе ошибок.

В соответствии с этими допущениям вероятность безотказной работы программы как функция времени t_i равна:

$$P(t_i) = e^{-\lambda_i t_i}, \lambda_i = C_D(N - (i - 1)), \qquad (15)$$

где i – число обнаруженных ошибок; C_D – коэффициент пропорциональности (по методу максимума правдоподобия $C_D >> 0{,}02$); N – первоначальное количество ошибок в программе.

В выражении (15) начальный момент времени соответствует моменту последнего $(i - 1 - \text{го})$ отказа программы.

Простая интуитивная модель. Применение этой модели предполагает, что тестирование проводится двумя группами программистов (или двумя

программистами в зависимости от сложности и объема ПС) независимо друг от друга, используя при этом независимые тестовые наборы. В процессе тестирования каждая группа регистрирует все найденные ими ошибки. При определении количества ошибок, оставшихся в программе, результаты тестирования обеих групп собираются и сравниваются. При этом будем считать, что первая группа обнаружила N_1 ошибок, вторая – N_2, а N_{12} – это число ошибок, обнаруженные обеими группами одновременно.

Обозначив через N заранее неизвестное количество ошибок, имеющихся в тестируемой программе до начала тестирования, можно оценить эффективность тестирования для каждой из групп E_1, E_2 следующим образом:

$$E_1 = N_1 / N, E_2 = N_2 / N \qquad (16)$$

Полагая вероятность обнаружения ошибок одинаковой для обеих групп, справедливо допустить, что если первая группа обнаружила определенное количество всех ошибок, она могла бы определить такое же количество для любого выбранного случайным образом подмножества. Данное допущение выразим математически:

$$E_1 = N_1 / N = N_{12} / N_2 \qquad (17)$$

Из формулы (16) следует, что $N_2 = E_2 N$, и после подстановки этого выражения в (17), получим: $E_1 = N_{12} / E_2 / N = N_1 N_2 / N_{12}$.

Модель Коркорэна. Она относится к статическим моделям надежности ПС, потому что в ней не фиксируются параметры времени тестирования, а учитывается количество N испытаний, в которых выявлено N_i ошибок i–го типа. В этой модели используются изменяющиеся вероятности отказов для различных типов ошибок. Так, выявление в ходе N испытаний ошибок - i-го типа происходит с вероятностью a_i, которая должна оцениваться на основе априорной информации о функционировании однотипных программных средств.

Модель Коркорэна позволяет получить оценку вероятности безотказной работы программы на момент проведения испытаний. При этом используется понятие уровень надежности R, который определяется по следующей формуле:

$$R = N_0 / N + \sum_{i=1}^{k} Y_i (N_i - 1) / N, \qquad (18)$$

где N_0 - число испытаний без отказов, выполненных в серии из N испытаний; k - известное число типов ошибок; Y_i - вероятность появления ошибок, равная:

$$Y_i = \begin{cases} a_i, N_i > 0 \\ 0, N_i = 0 \end{cases}.$$

Модель Шумана. Данная модель является динамической моделью дискретного времени, информация для которой собирается в процессе тестиро-

вания ПО в течение фиксированных или случайных интервалов времени. В модели используется предположение, что тестирование проходит в несколько этапов. На каждом этапе производится выполнение программы на полном комплексе разработанных тестовых данных, при этом выявленные ошибки регистрируются, но не исправляются. По окончании этапа производится расчет количественных показатели надежности, найденные ошибки исправляются, тестовые наборы изменяются и запускается следующий этап тестирования. Считается, что количество ошибок в программе постоянно и в процессе исправления программы новые ошибки не вносятся. Тогда скорость обнаружения ошибок считается пропорциональной количеству оставшихся ошибок.

Полагая, что на момент начала тестирования в программе присутствует E_t ошибок, а в течение времени тестирования τ обнаруживается ε_C ошибок в расчете на одну машинную команду, удельное число ошибок на одну команду машинного языка, оставшихся в системе после тестирования, составит:

$$\varepsilon_r = E_t \varepsilon_C(\tau) / I_t, \qquad (19)$$

где I_t - общее число машинных команд, которое считается постоянным в процессе всего этапа тестирования.

Считая, что частота отказов $Z(t)$ пропорциональна количеству ошибок, оставшихся в программе после затраченного на тестирование времени τ, получим

$$Z(t) = C\varepsilon_r(\tau), \qquad (20)$$

где C - некоторая постоянная; t - время работы программы без отказов.

В таком случае, если отсчитывать время работы программы без отказа t с момента $t = 0$, и интервал τ является фиксированным, то вероятность безотказной работы на интервале от 0 до t будет равна:

$$R(t,\tau) = \exp\left[C\left[E_t / I_t - \varepsilon_C(\tau)\right] \times t\right] \qquad (21)$$

А средняя наработка на отказ определится следующим выражением

$$t_{сз} = \frac{1}{C\left[E_t / I_t - \varepsilon_C(\tau)\right]} \qquad (22)$$

Определим начальное значение ошибок E_t и коэффициент пропорциональности - C.

Так как в ходе тестирования собираются данные о времени и количестве ошибок на каждом прогоне, т.е. общее время тестирования τ можно определить путем сложения времен каждого прогона:

$$\tau = \tau_1 + \tau_2 + \tau_3 + \ldots + \tau_n \qquad (23)$$

Предполагая, что интенсивность появления ошибок постоянна и равна λ, можно вычислить ее как число ошибок в единицу времени τ:

$$\lambda = \left(\sum_{i=1}^{k} A_i \right) / \tau \qquad (24)$$

При этом средняя наработка на отказ будет равна:

$$t_{cp} = \tau / \left(\sum_{i=1}^{k} A_i \right), \qquad (25)$$

где A_i - количество ошибок на i-м прогоне.

Имея сведения для двух различных моментов тестирования τ_a и τ_b, выбранных произвольно с учетом того, чтобы $\varepsilon_C(\tau_b) > \varepsilon_C(\tau_a)$, можно сопоставить уравнения (22) и (25) при τ_a и τ_b.

$$\lambda(\tau_a) = C\left[E_t / I_t - \varepsilon_C(\tau_a)\right] \qquad (26)$$

$$\lambda(\tau_b) = C\left[E_t / I_t - \varepsilon_C(\tau_b)\right] \qquad (27)$$

Тогда можно определить начальное значение ошибок коэффициент пропорциональности так:

$$E_t = \frac{I_t\left[\varepsilon_C(\tau_a)\lambda(\tau_b)/\lambda(\tau_a) - \varepsilon_C(\tau_b)\right]}{\lambda(\tau_b)/\lambda(\tau_a) - 1} \qquad (28)$$

$$C = \frac{\lambda(\tau_a)}{E_t - \varepsilon_C(\tau_a)} \qquad (29)$$

Получив значения E_t и C, можно рассчитать надежность программы по формуле (21).

Модель Миллса. Статистическая модель Миллса может быть использована не только для оценки количества ошибок до начала тестирования, но и для определения степени отлаженности ПО. До начала тестирования в программу преднамеренно вносятся ошибки, при этом считается, что обнаружение преднамеренно внесенных и собственных ошибок программы равновероятно.

Чтобы оценить число ошибок в программе до начала тестирования используется формула:

$$N = WS / V, \qquad (30)$$

где W - число ошибок, преднамеренно внесенных в программу до начала тестирования; V – число ошибок, обнаруженных в процессе тестирования из числа преднамеренно внесенных; S – количество «собственных» ошибок программы.

При продолжении тестирование до тех пор, пока все преднамеренно внесенные ошибки не будут обнаружены, можно оценить степень отлаженности программы C при помощи следующего выражения:

$$C = \begin{cases} 1, S > r \\ W/(W + r + 1), S \le r \end{cases}, \qquad (31)$$

где r - верхний предел (максимум) предполагаемого количества «собственных» ошибок.

Так как все преднамеренно внесенные ошибки считаются обнаруженными, то имеет место равенство значений $W = V$,.

Формулы (30) и (31) и представляют собой статистическую модель Миллса. Следует заметить, что если тестирование будет закончено преждевременно, то есть до того, как будут обнаружены все преднамеренно внесенные ошибки, то вместо выражения (31) можно использовать более сложное комбинаторное выражение (32), в котором полагается, что обнаружено только V ошибок из W преднамеренно внесенных:

$$C = \begin{cases} 1, S > r \\ \dfrac{\binom{W}{V-1}}{\binom{W+r+1}{r+W}} \end{cases}, \qquad (32)$$

где в круглых скобках записаны обозначения для числа сочетаний из W элементов по $V-1$ элементов и числа сочетаний из $W+r+1$ элементов по $r+V$ элементов в каждой комбинации.

Модель Миллса одновременно математически проста и интуитивно понятна. При ее использовании можно использовать специальную программу внесения ошибок, которая случайным образом выбирает модуль, вносит логическую ошибку, изменяя или убирая операторы, и затем заново его компилирует. Природа внесенной ошибки должна сохраняться в тайне, но все их следует регистрировать, чтобы впоследствии можно было разделять ошибки на исходные и внесенные.

2.7 Контрольные вопросы и задания

1. Как оценить ожидаемое число ошибок в программе, если использовать модель надежности программ на ранних этапах разработки?
2. Что представляют собой структурные схемы надежности с последовательным и параллельным соединением элементов? Как определяются для них показатели надежности?
3. Для чего используются модели надежности ПО ИС?
4. Какие параметры надежности ПО можно определить с помощью моделей надежности?
5. Какие модели надежности относятся к статическим, а какие к динамическим?
6. Каковы преимущества и недостатки известных моделей надежности?
7. С помощью каких моделей можно прогнозировать надежность ПО на этапах разработки ИС?
8. Какие существуют методы повышения надежности ПО?
9. Оцените общее число ошибок в тексте программы, если программа проверена тремя специалистами и если первый из них нашел в программе 3

ошибки, второй – 5 ошибок, а третий – 6 ошибок, причем две ошибки из найденных были общими у всех специалистов.

10. Напишите программу для обнаружения однократной ошибки на основе контроля четности

3 ОБЕСПЕЧЕНИЕ ПОКАЗАТЕЛЕЙ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Обеспечение качества ПО имеет особую значимость в связи с тем, что программные продукты приобрели статус коммерческих. Поэтому показатели качества могут включаться в контракты на разработку программных систем различного назначения. Обеспечение качества ПО - это планируемая систематическая совокупность действий для создания качественной продукции [8, 14, 15]. Рассмотрим далее способы обеспечения качества по отдельным показателям, в дополнение к рассмотренным в 2.5 способам создания надежного ПО.

3.1 Обеспечение функциональности

Функциональность программной системы (ПС) определяется функциональной спецификацией, основные положения которой задаются техническим заданием. Атрибутами, характеризующими этот показатель в соответствии с ГОСТ Р ИСО/МЭК 9126-93, являются *способность к взаимодействию, функциональная пригодность, защищенность и точность*.

Обеспечение функциональной пригодности. Функциональная пригодность ПС обеспечивается набором атрибутов качества, которые определяют назначение, основные, необходимые и достаточные функции ПС. Функционал системы задается на уровне требований техническим заданием в соответствии с потребностями заказчика или потенциального пользователя. При проектировании ПС свойства функциональной пригодности должны конкретизироваться в программных спецификациях в целом на систему и на компоненты. К ним относятся: функциональная полнота, степень покрытия функциональных требований, стабильность при модернизации ПС, количество реализованных требований заказчиков и другие.

В требованиях на ПС могут быть определены несколько уровней реализации функциональности ПС, в том числе может быть установлена некоторая упрощенная начальная или стартовая версия, которая подлежит реализации в первую очередь. После ее создания могут быть определены и последующие промежуточные версии. При этом возникает дополнительная технологическая задача, заключающаяся в организации наращивания функциональных возможностей ПС. Проектирование упрощенной версии ПС не следует считать созданием ее прототипа, ибо прототип создается для того, чтобы проанализировать и лучше понять условия применения будущей ПС [15] , а так-

же конкретизировать его внешнее описание. Упрощенная же версия ПС должна быть ориентирована на практически полезное применение теми пользователями, для которых оно предназначено. Тем не менее, основной принцип обеспечения функциональности заключается в том, чтобы ПС изначально разрабатывалась таким образом, как требуется от нее в полном объеме. Полный набор функциональных возможностей должен быть представлен сразу во внешнем описании и описании архитектуры ПС. Реализация же тех отдельных программных подсистем, определенных в архитектуре разрабатываемого ПС, функционирования которых не требуется в начальной версии, Может быть отложено в плановом порядке. При этом проектирование программных подсистем следует производить методом целенаправленного наращивания функциональности, используя в начальной версии ПС соответствующие имитаторы тех программных модулей, функционирование которых в этой версии пока не требуется. Возможна также упрощенная реализация некоторых программных модулей, в которой могут отсутствовать некоторых деталей выполняемых функций. В результате такой многоэтапной разработки ПС возникает проблема оценки ее завершенности.

Завершенность ПС является наиболее общей характеристикой качества ПС для выражения ее функциональности. Завершенность как характеристика качества является мерой того, насколько требуемая спецификация реализована в данной системе. Обеспечение этого свойства в полном объеме соответствует тому, что каждая функция, определенная в полной функциональной спецификации, реализована в полном соответствии со всеми указанными деталями и особенностями.

В следствии возникновения в проектируемой ПС ошибок достигнутая при обеспечении его функциональности завершенность по факту может быть не такой, как ожидалось. Поэтому следует считать, что завершенность достигнута с некоторой вероятностью, которая определяется объемом и качеством проведенного системного и модульного тестирования. Повышение этой вероятность может быть произведено в случае продолжения тестирования с последующей отладкой ПС.

Обеспечение способности к взаимодействию. Способность к взаимодействию (интероперабельность) ПО представляет собой возможность ПС взаимодействовать и функционировать с другими продуктами или системами без каких-либо ограничений доступа и реализации, в том числе:

- с операционной и аппаратной средой;
- с внешним окружением системы и ее пользователями;
- между отдельными внутренними компонентами ПС;
- между распределенными компонентами ПС.

Свойство интероперабельности обычно обеспечивается некоторым набором функций из состава специализированных библиотек, определённых в его внешнем описании и удовлетворяющих потребностям пользователей. Основными средствами обеспечения способности к взаимодействию являются:

- средства межпроцессных взаимодействий;
- механизм вызова удалённых процедур;
- архитектуры распределённых брокеров в технологиях CORBA, DCOM, JAVA и других.

Обеспечение точности. Обеспечение этого атрибута функциональности ПС обычно связано с математическими операциями над значениями вещественных типов, которые являются приближенными и представляются с некоторой погрешностью. Обеспечить необходимую точность при вычислении приближенного значения математического выражения или функции - значит необходимость получить это значение с такой погрешностью, которая не выходит за рамки допустимых пределов. Видами погрешностей приближенных вычислений, методами их оценки и методами достижения требуемой точности занимается вычислительная математика. Отметим при этом, что полная погрешность вычисляемого значения зависит:

- от погрешности используемого метода вычисления (в том числе неточность применяемой математической модели);
- от погрешности представления используемых данных, которая считается неустранимой вследствие ограничения на разрядность вещественных чисел;
- от погрешности округления, которая возникает при выполнении используемых в методе операций.

Обеспечение защищенности. Защищенность и безопасность функционирования представляют собой одну из наиболее трудно формализуемых характеристик сложных ПС, к числу которых нужно отнести следующие:

- соответствие установленным критериям и требованиям защиты от предумышленных и непредумышленных угроз безопасности ПС;
- соответствие методически верным методам и средствам защиты от проявления случайных ошибок в программах и данных;
- обеспечение эффективности оперативных методов защиты и восстановления при проявлениях угроз безопасности;
- соответствие нормативным документам и стандартам, регламентирующим способы защиты от различных типов угроз безопасности;
- обеспечение равнопрочности защиты и доступностью ресурсов для защиты.

К основным видам обеспечения безопасности ПС от искажения информационных и программных объектов относят защиты:

- от сбоев аппаратуры;
- от влияния «чужой» программы;
- от отказов «своей» программы;
- от ошибок пользователя;
- от несанкционированного доступа;
- от самой защиты.

Рассмотрим способы этих видов защиты программных средств далее подробнее.

Защита от сбоев аппаратуры. В последнее время этот вид защиты является не слишком злободневной задачей в связи с достигнутым уровнем надежности компьютерной техники. Такая защита обеспечивается организацией «двойных или тройных просчетов». При этом процесс обработки данных, реализуемый ПС, разбивается на заданные временные интервалы, разделенные «опорными точками». Длины таких интервала не должны превышать половины среднего времени безотказной работы аппаратуры компьютерного комплекса. В начале каждого такого интервала во вторичную память записывается копия состояния изменяемой в этом процессе памяти с некоторой контрольной суммой. Для того, чтобы удостовериться, что обработка данных за один «просчет» от одной опорной точки до следующей выполнена правильно, то есть без сбоев аппаратуры, производится два таких «просчета». После первого вычисляется и сохраняется полученная контрольная сумма, после чего состояние памяти восстанавливается по опорной точке и проводится второй «просчет» с новым определением контрольной суммы, которая сравнивается с результатом первого «просчета». Если обе эти контрольные суммы совпадают, то второй «просчет» считается правильным. Иначе контрольная сумма второго «просчета» также запоминается и производится третий «просчет», также с предварительным восстановлением памяти по опорной точке. Если третья контрольная сумма совпадет с контрольной суммой одного из первых двух «просчетов», то третий «просчет» считается правильным. Если же и это условие не выполняется, то фиксируется сбой аппаратуры, требующий инженерной проверки компьютерной системы.

Защита от влияния «чужой» программы. Необходимость такой защиты возникает при мультипрограммном режиме работы вычислительной системы, когда в памяти может одновременно находиться в стадии выполнения несколько программных процессов, которые попеременно получают процессорное управление в результате возникающих прерываний. Обработкой прерываний и управлением мультипрограммным режимом занимается операционная система. Под «чужой» программой понимается программа, выполняемая параллельно (точнее квазипараллельно) по отношению к защищаемой ПС. Необходимо обеспечить независимость выполнения защищаемой ПС от

программ, работающих параллельно с ней, в том числе и от функций операционной системы.

Можно выделить такие разновидности такой защиты как защита от отказов и защита от злонамеренного влияния «чужой» программы.

Защита от отказов посторонней программы означает, что на выполнение функций защищаемой ПС не будут влиять ошибки, возникающие в параллельно выполняемых программах. Для этого операционная система должна обеспечивать такие возможности как:

- защиту памяти;
- разные режимы функционирования компьютера: привилегированный и пользовательский;
- разные виды операций: привилегированные и ординарные;
- корректную реализацию прерываний и начального включения компьютера;
- временное прерывание.

Защита от злонамеренного влияния «чужих» программ означает, что изменение информационной среды защищаемой ПС со стороны параллельно выполняемой «чужой» программы становится невозможным или сильно затрудняется. Для этого операционная система должна располагать программными службами, обеспечивающими соответствующий контроль доступа к информационной среде выполняемых программных процессов. Обязательным условием такого контроля является обеспечение защиты от злонамеренного влияния «чужих» программ и самой операционной системы.

В этот вид защиты включается и защиту от «компьютерных вирусов», под которыми понимаются программы, способные в процессе своего выполнения внедряться путем копирования в другие и обладают способностью к размножению. При определенных условиях «вирусы» вызывают целенаправленное изменение хода выполнения «зараженной» программы, а это может привести к серьезным деструктивным изменениям всей ее информационной среды. Операционная система, если она сама защищена от влияния «чужих» программ, может ограничить доступ к программам, работающим в используемой информационной среде.

Защита от отказов «своей» программы. Данный вид защиты обеспечивается надежностью ПС и соответствующей организацией процесса проектирования. Эти вопросы были рассмотрены в разделах 2.4 и 2.5.

Защита от ошибок пользователя. В данном случае не имеется в виду защита от ошибочных данных, вводимых пользователем ПС. Такая защита связана с обеспечением устойчивости ПС, так как действия пользователя могут приводить к деструктивному изменению состояния информационной среды ПС, хотя вводимые им данные при этом могут быть корректны.

Защита от ошибок пользователя может обеспечиваться формированием и отображением предупредительных сообщений о попытках изменить состояние данных в системе с требованием подтверждения этих действий. В случае, когда такие ошибки все-таки совершаются, должна быть предусмотрена возможность восстановления состояния отдельных элементов информационной среды ПС на определенные предшествующие моменты времени. Для обеспечения такой возможности следует организовать ведение архива состояний информационной среды с учетом всех возникающих в ней изменений.

Защита от несанкционированного доступа. Отдельные пользователи ПС получают определенные, необходимые им для эффективной работы, информационные ресурсы и программные сервисы, при этом у разных пользователей ПС наборы этих ресурсов и сервисов могут отличаться, иногда весьма существенно. Данный вид защиты призван обеспечить возможность использования каждым пользователем ПС только тех функций, которые ему предоставлены. Для этого ПС в своей внешней информационной среде должна хранить информацию о пользователях и предоставленным им правах. Кроме того должны быть предоставлены определенные возможности пользователям для создания и редактирования этой информации. Защита от несанкционированного доступа к ресурсам и сервисам системы производится с помощью паролей, то есть секретных слов. Предполагается конечно, что каждый пользователь знает только свой пароль, который зарегистрирован в данной ПС этим пользователем. Для того, чтобы получить доступ к предоставляемым ему ресурсам и сервисам он и должен предъявить этот пароль.

Данный вид защиты имеет две разновидности: простая защита от несанкционированного доступа и защита от взлома защиты.

Простая защита от несанкционированного доступа обеспечивает защиту ресурсов и сервисов тем пользователям, которым не были предоставлены соответствующие права доступа или которые указывают неправильный пароль. Предполагается при этом, что получивший отказ пользователь не будет предпринимать попыток обойти или преодолеть эту защиту каким-то несанкционированным образом. Такой вид защиты целесообразно применяться и в ПС, , которые не обеспечивают полную защиту от влияния «чужих» программ при помощи средств операционной системы.

Защита от взлома защиты представляет собой особый вид защиты от несанкционированного доступа, цель которой заключается в создании существенных затруднений при преодолении этой защиты. Это необходимо для того, чтобы исключить возможные, часто настойчивые, попытки взломать или обойти защиту от несанкционированного доступа. Для этого следует предпринять дополнительные меры., которые состоят в том, что:

- необходимо обеспечить, чтобы такую защиту нельзя было обойти, т. е.

должна действовать защита от влияния «чужих» программ;

- необходимо усилить простую защиту от несанкционированного доступа применением специальных программистских приемов, которые затрудняют подбор подходящего пароля или его вычисление по информации, хранящейся во информационной среде ПС.

Защититься от этого можно так. Пароль X, то есть секретное слово или целое число, должен быть известен только владельцу защищаемых прав доступа и он не должен сохраняться во внешней информационной среде ПС. Для проверки прав доступа там должно хранится другое число $Y = F(X)$, которое однозначно вычисляется программой по предъявленному паролю X. Сама функция F может быть хорошо известной всем пользователям ПС, однако она должна обладать следующим существенным свойством: восстановление слова X по Y является практически невозможным. Такое число Y называется электронной подписью владельца пароля X и соответственно защищаемых прав доступа.

Еще один способ защиты от взлома связан с защитой сообщений, которые пересылаются по компьютерным сетям. Оно может представлять команду на дистанционный доступ к ценной информации, поэтому отправителю сообщения необходимо защитить его от возможных искажений. Применение компьютерной подписи в этом случае является недостаточным, так как защищаемое сообщение может быть перехвачено и подменено другим сообщением с сохранением и компьютерной подписи и пароля.

Для того чтобы защититься от такого взлома можно [14] использование наряду с функцией F, определяющей компьютерную подпись владельца пароля X, двух функции: *Stamp* и *Notary*. При передаче сообщения отправителю, помимо компьютерной подписи $Y = F(X)$, необходимо вычислить еще число $S = Stamp(X, R)$, где X пароль, а R - текст передаваемого сообщения. Здесь опять таки предполагается, что функция *Stamp* хорошо известна всем пользователям ПС и обладает свойством, что по S практически невозможно ни восстановить число X, ни подобрать другой текст сообщения R с заданной компьютерной подписью Y. Тогда передаваемое сообщение вместе со своей защитой должно иметь вид: (RYS). Еслии компьютерная подпись Y позволяет получателю сообщения установить истинность клиента, то S скрепляет защищаемый текст сообщения R с компьютерной подписью Y. Поэтому число S называют электронной печатью. Вторая функция *Notary(R,Y,S)* проверяет истинность защищаемого сообщения (R, Y, S) и это позволяет получателю сообщения однозначно установить, что текст сообщения R принадлежит владельцу пароля X.

Защита от защиты. Использование защиты от несанкционированного доступа может создать неблагоприятную ситуацию для самого владельца

прав доступа к ресурсам ПС. Например, если он забудет или потеряет свой пароль, не сможет воспользоваться своими правами. В таких ситуациях можно применять защиту от защиты. Для ее обеспечения ПС должно иметь привилегированного пользователя, который обычно называемого администратором ПС. Он должен, в частности, отвечать за функционирование защиты ПС: то есть он должен формировать контингент пользователей данного экземпляра Пси предоставлять каждому из этих пользователей определенные права доступа к ресурсам ПС. При этом ПС должна иметь привилегированную для администратора операцию, которая бы позволяла временно снимать защиту от несанкционированного доступа для каждого пользователя для того, чтобы изменения и зафиксировать новый пароль.

3.2 Обеспечение удобства применения

Одним из ключевых показателей качества ПО, определяющим его эргономичность, эффективность и привлекательность, является удобство использования, или практичность. Оно определяется такими свойствами, как понятность пользовательского интерфейса, легкость обучения, трудоемкость решения функциональных задач, производительность работы пользователя, частота появления ошибок, сбоев и жалоб на неудобства при использовании. Для того, чтобы создать действительно удобную программную систему необходим учет контекста их использования, знание психологии пользователей, использование справочно-обучающих средств, помогающих начинающим пользователям и предоставляющим нужные сведения для эффективной работы опытных. Самым же значимым фактором удобства применения является эффективность решения задач пользователей с помощью соответствующей программной системы.

Большое значение при этом имеют *психологические и физиологические факторы*. Теоретической основой для определения степени удобства и неудобства участия человека в функционировании и способах использования человеко-машинного комплекса является когнитивная психология, в задачу которой входит изучение познавательных процессов человеческого сознания. В данном случае мы имеем дело с инженерной психологией, то есть психологией применения в процессах практической деятельности человека машин, инструментов, оборудования и приборов различного назначения.

В задачах проектирования удобного в использовании ПО используется большое количество информации, среди которой наиболее важные для разработки пользовательского интерфейса сведениями являются данные, характеризующие:

- свойственность человеку ошибаться;
- скоростные показатели деятельности пользователя при работе с

программой и при выполнении должностных обязанностей;

- внимание человека;
- понятность программного интерфейса и принципов использования ПО;
- память человека;
- различие категорий пользователей.

Все эти факторы удобства использования в основном определяют принципы проектирования удобного пользовательского интерфейса. Он представляет собой комплекс программных и аппаратных средств, используемых для эффективного взаимодействия пользователя с компьютерной системой. Одним из ключевых элементов такого взаимодействия являются *диалог,* под которым в контексте взаимодействия человека с машиной *понимают регламентированный процесс обмен информацией* между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное и эффективное решение выполняемой задачи. Диалог состоит из некоторой совокупности отдельных операций ввода / вывода, которые по заданному алгоритму физически обеспечивают связь пользователя и программной системы, использующей компьютерную технику. Как правило, обмен информацией при этом осуществляется передачей сообщений, команд или инструкций.

Разработчикам интерфейсов необходимо учитывать как физические, так и умственные способности людей, которые будут использовать данное ПО. Для этого следует руководствоваться целым рядом принципов, позволяющих обеспечить удобство использования ПО.

Принцип учета знаний пользователя предполагает требование к интерфейсу быть настолько удобным в эксплуатации, чтобы пользователям не понадобились особые усилия для привыкания к нему. При этом следует использовать термины, понятные пользователю. Объекты, управляемые ПС, должны быть прямо связанные с рабочей деятельностью пользователя. Например, если создается система, предназначенная для авиадиспетчеров, то объектами управления в ней должны быть самолеты, взлеты и посадки, сигнальные знаки и т.п. Особенности системной реализацию интерфейса в терминах файловых структур и структур данных следует скрывать от пользователя.

Принцип согласованности интерфейса пользователя означает, что команды и меню системы должны иметь одинаковый формат, параметры следует передавать во все команды одинаково с сохранением похожей пунктуации команд. Такие интерфейсы позволяют существенно сократить время на обучение пользователей, причем навыки, полученные при изучении какой-то команды или отдельного элемента приложения, должны быть потом исполь-

зованы при работе с другими частями ПС.

Принцип минимума неожиданностей предполагает, что количество неожиданностей должно быть как можно меньшим, так как пользователей раздражает, если система вдруг начинает себя вести непредсказуемо. Поскольку при работе с программной системой у пользователей формируется специфичная модель ее функционирования, то если действие пользователя в некоторой ситуации вызывает определенную реакцию системы, естественно ожидать, что такие же действие в другой ситуации приведут к аналогичной реакции. Поэтому разработчики пользовательских интерфейсов должны обеспечивать диалог так, чтобы похожие действия создавали похожий эффект.

Очень важным является *принцип восстановления системы*, потому что всегда имеется вероятность совершения пользователем ошибки. Грамотно и правильно разработанный интерфейс позволяет уменьшить количество ошибок пользователя. Так, применение меню позволяет избежать ошибок, которые могут появиться при вводе данных с клавиатуры. Тем не менее полностью устранить возможность появления ошибки невозможно, поэтому интерфейсы должны иметь средства защиты от ошибок пользователя, а также средства, которые позволяют корректно восстановить информацию после ошибок.

Принцип поддержки пользователя, которая должна заключаться во включении в интерфейсе средств поддержки пользователей для обеспечения разных уровней помощи и необходимой справочной информацией. Справочная информация должна иметь несколько уровней – от основ, нужных начинающим пользователям, до полного описания возможностей ПС. Справочная система должна иметь иерархическую структуру, не допуская перегруженности пользователей лишней информацией при простых запросах.

Принцип учета разнородности пользователей предполагает, что с ПС могут работать разные типы пользователей. Разные группы пользователей по разному ее используют. Так, некоторые пользователи работают с системой нерегулярно, время от времени, другие, "продвинутые пользователи", используют данное ПО каждый день по несколько часов. «Редкие пользователи» нуждаются в интерфейсе, который направлял их работу с системой, в то время как опытным необходим интерфейс, который позволил бы им максимально быстро взаимодействовать с ПС, решая практические задачи. Следует иметь в виду, что некоторые пользователи могут иметь разные физические ограничения, поэтому в интерфейсе должны быть средства, которые бы позволяли произвести настройку интерфейс под индивидуальные особенности отдельного человека. Сюда можно включить средства, позволяющие увеличивать отображение текста, замещать звук текстом и наоборот, создавать кнопки увеличенных размеров и т.п.

Основными факторами, с помощью которых можно оценить, а иногда и измерить показатель удобства использования ПО, являются:

1) *адекватность интерфейса* , то есть его соответствие задачам, с которыми пользователи должны и хотели бы работать;

2) *производительность работы* пользователей – это скорость решения однотипных задач при помощью конкретного ПО;

3) *скорость обучения* новых пользователей, которая оценивается как количество задач, выполнению которых новый пользователь самостоятельно обучается за некоторый промежуток времени;

4) *эффективность предотвращения и преодоления ошибок* пользователей;

5) *субъективное удовлетворение* пользователей при работе с системой.

Специалисты по проектированию пользовательских интерфейсов используют определенный набор правил, позволяющих как оценивать качество пользовательского интерфейса и генерировать проектные решения, повышающие это свойство. К ним относятся:

- правило доступности;
- правило эффективности;
- правило непрерывного развития;
- правило поддержки;
- правило соблюдения контекста.

Поиск эффективных решений, повышающих качество пользовательского интерфейса, позволяет осуществить использование следующих принципов:

- принцип структуризации;
- принцип простоты;
- принцип видимости;
- принцип обратной связи;
- принцип толерантности;
- принцип повторного использования.

Рассмотрим далее методы проектирования удобного в использовании ПО. Одним из известных подходов к созданию удобного пользовательского интерфейса является проектирование, ориентированное на использование (*usage-centered design*), принципы которого заложены Л. Константайном и Л. Локвудом [16]. Основная идея этого метода заключается в использовании специальных моделей, которые способствуют определению адекватного набора задач, доступного для решения пользователям, а также способов организации данных, ориентированных на более простое их решение.

В рамках проектирования, ориентированного на использование, может быть использовать следующий набор моделей.

1) *Модель ролей.* Эта модель представляет собой перечень и содержание ролей пользователей ПС. Под ролью понимается группа связанных для некоторого множества пользователей задач, функций и потребностей. Такая модель позволяет определить связи между ролями, например, роли могут уточнять друг друга, включаться друг в друга или просто быть похожими. Кроме того, можно установить набор из нескольких центральных ролей, на которые следует направить основное проектирование.

2) *Модель задач.* Эта модель пользовательского интерфейса формируется на основе сущностных вариантов использования (*essential use cases*). Сущностный вариант использования отличается от обычного тем, что в рамках его сценариев выделяются только цели и задачи пользователя, конкретные действия не включаются. В таком виде модель задач представляет собой граф вариантов использования со связями между ними по обобщению, расширению и использованию. Многие из моделируемых вариантов использования могут быть определены как основные – без них ПС лишится значительного числа пользователей.

3) *Модель содержимого.* Эта модель описывает набор взаимосвязанных контекстов взаимодействия и рабочих пространств, к которым относятся экраны, формы, окна, диалоги, страницы и прочие элементы, с входящими в них данными и возможными действиями. При формировании модели содержимого следует определять, какие данные и функции войдут в состав пользовательского интерфейса, а не решать вопрос того, как это рабочее пространство будет выглядеть.

После того, как набор контекстов с их информационным и функциональным содержимым будет определен, создается карта навигации между контекстами, в которой отображаются возможные переходы между ними. Эта карта должна объединить различные контексты взаимодействия в рамках единой модели содержимого.

Чтобы можно было осуществлять контроль удобства использования ПО применяются различные виды инспектирования. В частности, распространено э*вристическое инспектирование* [16], которое формируется как систематическая оценка элементов и аспектов интерфейса с точки зрения различных эвристик. В качестве последних можно применять приведенные выше правила и принципы построения практичных и эргономичных интерфейсов, а также другую достаточно полную систему правил и принципов, которые обычно приводятся в методических руководствах по удобству использования ПС.

В рамках определенного сеанса инспектирования несколькими специалистами, имеющими опыт в деятельности такого род, проводится оценка интерфейса. Число таких экспертов может составлять от 3 до 5 человек. Индивидуальные результаты обобщаются и объединяются в общую картину. Не-

посредственно в процессе инспектирования проектировщики должны отвечать на различные вопросы, относящиеся с аспектам как предметной области, так и функционированию анализируемой ПС. Обычно такая проверка проводится в два этапа. Сначала исследуется архитектура пользовательского интерфейса в целом, а затем отдельные аспекты взаимодействия и используемые его элементы.

При контроле пользовательского интерфейса можно использовать такую характеристику как сущностная эффективность, под которой понимается степень приближения производительности пользователей к некоторой идеальной величине при работе с данным интерфейсом. Она оценивается процентным отношением количества действий, выполняемых пользователем при наиболее эффективном (идеальном) варианта использования, к количеству действий пользователя, взаимодействующего с анализируемой ПС.

Важной характеристик удобства использования является *согласованность задач*, вычисляемая следующим образом:

1) выполняемые пользователем задачи располагаются в порядке убывания частоты их возникновения;

2) определяется количество действий пользователя, выполняемых для решения каждой задачи;

3) вычисляется индекс согласованности, для этого по каждой паре задач производится следующий подсчет: если порядок в этой паре по трудоемкости выполнения совпадает с порядком по частоте использования, к индексу прибавляется 1, иначе – из индекса вычитается 1;

4) рассчитывается итоговая согласованность задач как процентное отношение индекса согласованности к общему количеству различных пар задач.

Кроме перечисленных методов оценки удобства применения применяется также тестирование удобства использования, как правило, на поздних стадиях проектирования.

3.3 Обеспечение сопровождаемости

Сопровождение (поддержка) ПО представляет собой процесс улучшения, оптимизации и устранения дефектов в нем после передачи в эксплуатацию. Сопровождение ПО является одной из фаз жизненного цикла программного продукта, которая следует за фазой передачи в эксплуатацию. В процессе сопровождения в программу вносятся изменения, направленные на исправление обнаруженных в процессе эксплуатации дефектов и недоработок, а также для добавления новой функциональности, с целью повысить удобство использования и функциональность ПО.

Процесс сопровождения обуславливает специфические технические и требования по управлению персоналом, занятым поддержкой ПО. Связанные

с этим проблемы можно представить следующими категориями:

- технические проблемы;
- управленческие проблемы;
- проблемы оценки стоимости;
- проблемы измерения.

К техническим проблемам относятся:

- проблема *ограничения понимания*, которые подразумевает, насколько быстро специалист по сопровождению может понять, в чем заключается проблема и где необходимо внести изменения в код системы;
- проблема *тестирования* связана с тем, что стоимость проведения полного набора тестов для программных модулей системы может быть существенной как по времени, так и по стоимости;
- проблема *анализа влияния* состоит в необходимости полного анализа возможных последствий и влияний изменений, вносимых в существующую систему;
- проблема *возможности сопровождения* как анализ наличия и оценки применимости методов сопровождения для выполнения заданных требований;
- проблема *отсутствия системной документации* (системной модели) мешает формированию понимания системы и, как следствие, возможности адекватного анализа;

Технические проблемы могут быть решены на основе систематического подхода к процессам проектирования и сопровождения, применению соответствующих технологий и средств автоматизации, используемых при решении задач по поддержке жизненного цикла с помощью специализированных инструментальных средств.

Управленческие проблемы включают в себя:

- проблему *согласования с организационными целями* состоит в том, что цели проектирования и сопровождения принципиально отличны;
- проблемы *кадрового обеспечения*, которые связаны с вопросами привлечения и удержания квалифицированного персонала по сопровождению;
- проблему *уникальных работ* в процессе связана с тем, что сопровождение включает процедуры, не представленные в процессе разработки, что требует от менеджмента специального внимания и специальных знаний;
- проблему *организации сопровождения* состоит в необходимости решения вопросов ответственности за конкретные функции деятельности по сопровождению;
- проблему *аутсоурсинга*, которая связана с необходимостью передачи работ другим компаниям.

Кроме того, важной проблемой является *оценка стоимости работ* по сопровождению ПО. Для оценки стоимости сопровождения используют:

- методы использования параметрических моделей, использующих данные предыдущих проектов;
- методы экспертного мнения, позволяющие повысить точность оценок, полученных при использовании параметрических моделей за счёт применения опыта.

Самые хорошие результаты получаются когда эмпирические методы сочетаются с накопленным опытом эксплуатации. Получаемые данные представляют собой результаты оценки основных аспектов сопровождения.

Проблемы *измерения в сопровождении* выражаются в том, что данные измерений показателей качества объединяются в единую систему количественных оценок, характеризующих данное ПО. Для решения этих проблем многие организации и предприятия используют современный и популярный подход, базирующийся на оценке количества проблем и статуса их решений. Методология такого подхода определена в проекте *Practical Software and Systems Measurement* (PSM).

В последнее время в программной инженерии широкое распространение получило конфигурационное управление (*software configuration management*, SCM), которое представляет собой комплекс методов, направленных на систематический учёт изменений, вносимых разработчиками в программный продукт в процессе его разработки и сопровождения, сохранение целостности системы после изменений, предотвращение нежелательных и непредсказуемых эффектов, формализацию процесса внесения изменений.

Действия по сопровождению включают в себя гораздо больше, нежели просто технические изменения и дополнения. Для отражения каждого такого действия требуется обновление всей цепочки документации. Кроме того, необходимо изменение состояния системы управления конфигурациями для отражения новой версии продукта. Если игнорировать необходимость обновления, то документация потеряет согласованность, из-за чего стоимость сопровождения начнет возрастать и, в конце концов, окажется просто неприемлемой.

Обеспечение сопровождаемости сводится к обеспечению таких свойств как изучаемость и модифицируемость ПО.

Изучаемость программы определяется прежде всего составом и качеством документации по сопровождению и выражается через такие характеристики качества как документированность, информативность, понятность, структурированность удобство восприятия. Последние два свойства, а также понятность связаны с изучением текстов программных модулей.

При окончательном оформлении текста программного модуля следует

придерживаться следующих рекомендаций, которые определяют наиболее эффективный с точки зрения читаемости стиль программирования:

- необходимо использовать в тексте модуля комментарии, проясняющие и объясняющие особенности принимаемых решений; включать комментарии следует на самой ранней стадии разработки модуля;
- нужно использовать осмысленные и устойчиво различимые имена при длине имени от 4 до 12 литер, цифры должны располагаться в конце, не следует использовать единый стиль при определении имен и ключевых слов;
- следует соблюдать осторожно использовать константы, так как уникальная константа должна иметь единственное вхождение в текст модуля;
- не стоит бояться использовать необязательные скобки, потому что они обходятся дешевле, чем ошибки;
- следует в строке размещать не более одного оператора;
- для того, чтобы структура модуля была наиболее понятной, надо использовать дополнительные пробелы и отступы в начале каждой строки, при этом улучшается читаемость текста модуля;
- необходимо избегать трюков, а именно таких приемов программирования, при которых создаются фрагменты модуля, основной эффект которых не очевиден или скрыт, при этом могут возникать побочные эффекты реализации программных функций.

Для того, чтобы создать качественное оформление текста программы следует применять некоторые принципы и паттерны проектирования [16]. К таким принципам относятся.

Принцип *единственной обязанности*, заключающийся в том, что у класса может быть только одна причина для изменения. Поэтому классу следуетпоручать только одну обязанность.

Принцип *открытости/закрытости* устанавливает необходимость программных элементов, таких как классы, модули и функции, быть открыты для расширения, но закрыты для изменения.

Принцип *подстановки* определяет наличие возможности вместо базового класса подставлять любой его подтип.

Принцип *инверсии зависимостей* заключается в том, что модули верхнего уровня не могут зависеть от модулей нижнего уровня, однако и те и другие должны зависеть от абстракций, а также что абстракции не должны зависеть от деталей, наоборот, детали должны зависеть от абстракций.

Принцип *разделения интерфейсов* устанавливает обязанность клиентов знать только об абстрактных интерфейсах, обладающих свойством сцепления. Сцепление модулей является мерой взаимной зависимости модулей по данным и определяется как способом обмена данных между модулями, так и свойствами передаваемых данных.

Паттерны проектирования предлагают универсальные, проверенные практикой решения. Среди обширного перечня различных паттернов следует выделять и использовать те, которые рекомендуется применять при гибкой разработке программного обеспечения. При этом использование паттернов при разработке позволяет разрабатывать ПО, которое можно легко модифицировать и сопровождать.

Модифицируемость программы определяется в основном свойствами документации, и свойствами, которые реализуются программным путем, она выражается через такие характеристики качества как расширяемость, модифицируемость, структурированность и модульность.

Расширяемость определяется возможностями автоматической настройки на условия применения программы по информации, которая задается пользователем. К этим условиям относятся: конфигурация компьютера, на котором устанавливается ПС, в том числе объем и структура его памяти, требования конкретного пользователя к функциональным возможностям, такие как требования к режиму применения программы, требования к конкретной структуре информационной среды. Кроме того, сюда же можно отнести и возможность добавления в ПС определенных компонент. Для того, чтобы можно было реализовать возможности расширения ПС часто включает дополнительную подсистему, которая называемая инсталлятором. Он осуществляет прием от пользователя необходимой информации и настройку программы в соответствии с принятыми требованиями. Решение о включении в ПС инсталлятора принимается в процессе разработки ее архитектуры.

Модифицируемость обеспечивается приданием ПС таких свойств документации и программных модулей, которые облегчают внесение изменений и доработок в документацию и программы как вручную, так и с определенной компьютерной поддержкой. В техническом задании на ПС могут быть указаны некоторые приоритетные направления и особенности развития ПО, которые должны быть учтены при разработке архитектуры системы и ее модульной структуры. Ручная модификация программных компонент упрощается при обеспечении таких свойств как структурированность и модульность.

Еще одна наиболее общая проблема сопровождения ПС заключается в обеспечении возможности всех ее компонент оставаться согласованными в каждой новой версии программы. Обычно этот процесс называют управлением конфигурацией (*configuration management*).

3.4 Обеспечение мобильности

Необходимость в обеспечении мобильности возникает вследствие того, что быстрое развитие аппаратных средств компьютерной техники делает жизненный цикл многих больших программных систем намного продолжительнее, чем период оправданного эффективностью использования применения аппаратных комплексов, для которых они первоначально создавались. Продление жизненного цикла таких ПС путем перехода на более современные аппаратные платформы и делает решение проблемы мобильности весьма важной задачей.

Понятие мобильности включает в себя свойство *кроссплатформенности*, под которым понимается способность ПО работать более чем на одной аппаратной платформе и (или) операционной системе (ОС). Обеспечивается кроссплатформенность благодаря использованию высокоуровневых языков программирования, сред разработки и выполнения, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ. Типичным примером является ПО, предназначенное для работы в разных ОС одновременно.

Кроме того, мобильность ПО определяется такими свойствами как:

- переносимость ПО между различными аппаратными платформами, в первую очередь характеризуемыми архитектурой процессора;
- переносимость ПО между различными ОС;
- комбинация вышеперечисленных вариантов;
- переносимость между различными версиями одной операционной системы;
- переносимость между программными окружениями промежуточного уровня.

При переноса программ возникают следующие задачи и соответствующие им процессы:

- обеспечение сохранности инвестиций, вложенных в уже реализованные и апробированные ПС и базы данных при условии увеличения ресурсов и функций аппаратных комплексов и операционных сред;
- снижение трудоемкости, стоимости и продолжительности непосредственной разработки сложных распределенных ПС и баз данных;
- обеспечение оптимального качества, надежности и безопасности функционирования ПС;
- обеспечение возможности эффективного переноса апробированных программных продуктов на различные ОС и аппаратные платформы с минимальными издержками;

- эффективная реализация совместной работы и интероперабельности ПС при взаимодействии с другими программами в рамках решения единой целевой задачи на различных локальных и распределенных платформах;

- обеспечение мобильности пользователей путем унификации стилей взаимодействия ПС с пользователями

- снижение зависимости заказчиков конкретных ПС от поставщиков аппаратных и операционных платформ и от разработчиков ПО.

При оценке мобильности ПО основными объектами анализа являются:

- программные модули и функциональные компоненты ;
- готовые программные продукты и пакеты прикладных программ;
- крупные программные комплексы определенного функционального назначения;
- системы управления базами данных;
- информационные массивы баз данных;
- электронные документы, описывающие программы и данные.

Мобильность ПС определяется такими атрибутами качества как независимость от устройств, автономность, структурированность и модульность.

Чтобы ПС обладала свойствами независимости от устройств и автономности, а ее программные модули были бы представлены на машинно-независимом языке программирования, перенос ПС в другую программно-аппаратную среду необходимо обеспечивать перекомпиляцию программ в этих среде. В настоящее время все еще незначительная часть реальных систем обладает таким качеством. Такими свойствами проще обеспечить отдельные программные модули больших систем, в том числе и весьма значительные. Это обстоятельство и определяет тот наиболее рациональный путь, каким следует добиваться мобильности ПС.

Если свойства ПС зависят от аппаратуры, то в техническом задании должна быть описана эта компьютерно-аппаратная среда (аппаратной платформа) [15, 16]. Устранить эту зависимость можно обеспечением автономности ПС. Поскольку ПС строится на базе некоторой операционной системы (ОС), которая скрывает специфику аппаратной платформы, делая ПС независимым от устройств. В этом случае в техническом задании должна быть описана эта программная операционная среда с указанием промежуточного ПО, обеспечивающего такую автономность. Таким образом можно отметить, что мобильность ПС будет непосредственно связано с мобильностью используемой ОС. При этом перенос ПС на другую аппаратную платформу может осуществляться автоматически, если на эту платформу будет осуществлен перенос используемой ОС. Однако, такое обеспечение мобильности непосредственно ОС является самостоятельной и довольно трудной задачей.

Приведенные обстоятельства объясняют, что для обеспечения мобильности ПС необходимо решить две задачи:

1) выделить по возможности большую части программ ПС, которые бы обладали свойствами независимости от устройств и автономности, то есть были бы независимы от аппаратно-операционной платформы;

2) обеспечение сопровождаемости для оставшихся частей программ ПС.

Для того, чтобы решить эти задачи следует выбрать в качестве архитектуры ПС слоистую систему. В ней должен быть сформирован основной слой, реализующий основные функции ПС, он должен быть максимально независимым от аппаратно-операционной платформы. Кроме того надо выделить также слой, называемый часто ядром, в который надо включить программные модули, зависящие от аппаратно-операционной платформы. Этот слой также должен обеспечивать доступ к внешней информационной среде ПС. Между этими слоями необходимо определить *системный интерфейс*, независимый от аппаратно-операционной платформы, при помощи которого обеспечиваются обращения из основного слоя к модулям ядра и обратно. В случае, если ПС требует использование графических пользовательских интерфейсов, то необходимо выделение еще одного программного слоя, зависящего от той части аппаратно-операционной платформы, которая средства для создания таких интерфейсов. Такой слой называют *графической оболочкой*. Между графической оболочкой и основным слоем также необходим системный интерфейс, обеспечивающий правила обращения из оболочки к модулям основного слоя и не зависящий от пользовательской платформы.

Проектирование модульной ПС позволяет сформировать указанные слои, выделяя программные модули с требуемыми свойствами и распределяя их между указанными слоями. Также модульность и структурированность графической оболочки и ядра позволяют придать этим слоям свойство модифицируемости. Желательно при этом обеспечить, чтобы каждый модуль оболочки и ядра был ориентирован на реализацию конкретных функций управления четко выделенной компоненты аппаратно-операционной среды, для чего рекомендуется использовать такие методы как унификация интерфейсов, стандартизация протоколов и другие. [16].

Чтобы создавать переносимые программы, необходимо использовать следующие подходы.

1) *Переиспользование бинарных файлов*. Перенос приложения на новую программно-аппаратную платформу осуществляется без больших проблем для разработчиков, если старая и новая версии ПС совместимы на бинарном уровне, что означает поддержку новой системой двоичного интерфейса приложений ABI (Application Binary Interface).

Основными составляющими ABI являются:

- форматы исполняемых файлов и библиотек;
- набор библиотек и их функций, предоставляемых системой.

2) *Переиспользование исходного кода,* что означает использование одного и того же исходного кода для сборки приложения на различных аппартно-операционных платформах. Такая возможность обеспечивается наличием использованием в целевых системах компиляторов для соответствующего языка программирования, а также наличием и доступность необходимых библиотек.

3) *Использование интерпретируемого кода.* Этот подход предполагает запись программного кода на интерпретируемых языках, применение которых не предполагает создания исполняемых файлов в формате целевой ОС, так как интерпретатор кода последовательно считывает и выполняет инструкции непосредственно из текста программы. Такая прямолинейная интерпретация однако является не эффективной, так как у интерпретатора практически отсутствуют возможности для оптимизации кода.

4) *Использование эмуляторов ABI.* ABI представляет собой набор соглашений для доступа приложения к ОС и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами. В отличие от интерфейса прикладного программирования API (*application programming interface*), который регламентирует совместимость на уровне исходного кода, ABI можно рассматривать как набор правил, позволяющих компоновщику объединять откомпилированные модули без перекомпиляции всего кода. В ряде случаев сама ОС может обеспечивать бинарную совместимость с другой системой при помощи специального дополнительного слоя совместимости. Существует также продукты разных разработчиков, позволяющие загружать файлы других ОС посредством использования транслятора ABI, загружающего файлы требуемого формата, преобразуя вызовы функций и процедур, определенные внутри файла, в соответствующие вызовы текущей ОС.

5) *Виртуализация.* Представляет собой альтернативу эмуляции ABI. Виртуализация предполагает запуск внутри основной ОС копии системы, которая с использованием программ эмулирует нужное аппаратное окружение. Такая программная эмуляция получила название *виртуальной машины*. На ней может быть установлена ОС и другое окружение, необходимое выполняемому приложению, хотя само это приложение запускается уже в родной для него среде.

6) *Использование Web-технологий*. Это наиболее современный подход, позволяющий строить распределенные системы, то есть системы, компоненты которых располагаются на разных компьютерах с разными апаратно-опе-

рационными платформами. Выделяются два основных способа построения таких приложений с использованием Web-технологиq:

- создание локального приложения, то есть программы, расположенной на машине пользователя, при этом взаимодействие с другими частями ПС обеспечивается использованием web-браузера;
- создание приложения в виде сервиса; при этом серверная и клиентская части приложения могут быть размещены на разных машинах.

При проектировании распределенной системы используются такие механизмы взаимодействия компонентов как:

- спецификация CORBA (*Common Object Request Broker Architecture*),
- протоколы обмена сообщениями на базе расширяемого языка разметки XML (*eXtensible Markup Language*);
- парадигма сервисно-ориентированной архитектуры SOA (*Service-Oriented Architecture*) используется при проектировании сложных распределенных автоматизированных систем, при этом программный комплекс часто реализуются как набор Web-сервисов.

3.5 Обеспечение эффективности ПО

Эффективность ПС обеспечивается в ходе принятия многочисленных проектных решений на разных этапах ее разработки, начиная с создания программной архитектуры. Сильное влияние на эффективность ПС, особенно в отношении к используемым ресурсам, таким как процессор и память, оказывает выбор структуры и представления данных. Не менее сильно зависит эффективность, особенно с точки зрения скорости обработки информации, и выбор алгоритмов, которые используются в составляющих систему программных модулях, в том числе и особенности их реализации, включая выбор языка программирования, среды программирования и других факторов. Таким образом, при обеспечении эффективности ПС постоянно приходится разрешать противоречие между *временно́й эффективностью* и *эффективностью по ресурсам*, в том числе по памяти. Важным обстоятельством при этом имеет указание в техническом задании или спецификации качества приоритетов или количественного соотношения между этими атрибутами эффективности ПО. Нужно отметить, что составляющие систему программные модули по-разному влияют на эффективность ПС в целом: одни могут определять временные параметры и слабо зависеть от используемых ресурсов, а другие наоборот могут не оказывать заметного влияния на время работы ПС, но существенно влиять на общий расход заминаемой памяти и занятость процессора. Эти обстоятельства определяют, что в целом эффективность ПО далеко не всегда можно правильно оценить, особенно на тех этапах

проектирования, которые собственно и определяют основные параметры быстродействия и потребность в ресурсах вычислительной системы .

На основе приведенных факторов можно определить основные принципы, приведенные ниже, которыми рекомендуется руководствоваться для обеспечения эффективности ПС [15, 16]:

- прежде всего следует разработать надежное ПС, и только потом заниматься доведением его эффективности до требуемого уровня в соответствии с требованиями;
- обеспечить высокую эффективность поможет использование оптимизирующего компилятора;
- в случае, когда эффективность ПС не удовлетворяет требованиям качества, необходимо найти наиболее критические модули, снижающие эффективность по времени и по ресурсам и попытаться в первую очередь оптимизировать их путем ручной переделки;
- оптимизацией программных модулей, которые не создают проблем в точки зрения эффективности ПС, не стоит заниматься, по крайней до тех пор, пока не оптимизированы критические модули.

Для того, чтобы найти критические модули, определяющие временной эффективности ПС, нужно получить распределение по модулям времени работы ПС путем соответствующих измерений в процессе тестирования или при эксплуатации ПС. Существуют также специальные программные инструменты, динамические анализаторы, которые позволяют определить частоту обращения к каждому модулю в процессе применения ПС.

3.6 Контрольные вопросы и задания

1) Что понимается под завершенностью программной системы?
2) Какими основными аспектами характеризуется атрибут защищенности ПС?
3) Какие принципы позволяют обеспечить удобство использования ПО?
4) Дайте сравнительную оценку различных подходов к обеспечению мобильности ПС с целью их применения для разработки конкретной ПС?
5) Какими факторами определяется модифицируемость ПС?
6) Как понимается практически оправданный стиль программирования, чем он определяется?
7) Каким образом использование Web-технологий позволяет обеспечить высокую мобильность ПС?
8) Как модульность ПС влияет на временную эффективность и эффективность по ресурсам?

4 ОЦЕНКА КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Как уже отмечалось, в системе управления качеством программных систем (ПС) ключевое место занимает процесс измерения и сбора информации об их состоянии с точки исследования и анализа характеристик качества. Общая модель системы оценивания качества ПС показана на рисунке 4, а эффективность этой модели и достоверность получаемых результатов будет определяться совокупностью методов определения показателей, их атрибутов и метрик.

4.1 Методы оценки показателей качества ПО

Существуют следующие группы методов сбора данных при исследовании различных систем, которые используются и для определения показателей качества ПО:

- по способам получения информации различают органолептические, расчетные, измерительные и регистрационные методы;

- по источникам получения информации могут быть методы традиционные, экспертные и социологические.

В *измерительном методе* получение сведений о свойствах и характеристиках ПО осуществляется с использованием инструментальных средств. С использованием этого метода можно определять временные и объемные характеристики ПО, а также количественных показатели, такие как число исполненных операторов, число ветвей в программе, число точек входа (выхода) и другие.

Регистрационный метод базируется на фиксации данных во время испытаний или эксплуатации ПО, когда регистрируются, учитываются и подсчитываются определенные события, например, время и число сбоев и отказов, время передачи управления другим модулям, время начала и окончания работы.

Органолептический метод представляет собой способ получения информации в результате анализа восприятия человеческих органов чувств (зрения, слуха), и применяется для определения таких качественных характеристик таких показателей как удобство применения, эффективность и т. п.

Расчетный метод использует вычисления по теоретическим или эмпирическим зависимостям характеристик качества от статистических данных, накапливаемых при испытаниях, эксплуатации и сопровождении ПО. примерами расчетного метода можно считать точность вычислений, время реакции, объем необходимые ресурсы так далее.

Экспертный метод используется, когда характеристика качества не может быть получена выше перечисленными способами, или их использова-

ние являются существенно более трудоемким. Определение значений при этом осуществляется группой экспертов-специалистов, компетентных в решении данной задачи, на базе их опыта и интуиции.

Социологические методы оценки характеристик качества основаны на обработке специальных анкет-вопросников.

Как уже отмечалось, характеристики качества объединены в иерархическую систему из четырех уровней, при этом каждый показатель вышестоящего уровня определяется по связанным с ним количественным характеристикам нижестоящих уровней.

Такая система обеспечивает возможности получения интегральной оценки по иерархическим группам характеристик качества, в в которые входят показатели качества (1-й уровень): атрибуты показателе (2-й уровень), метрики для оценки атрибутов (3-й уровень) и, наконец, оценочные элементы как единичные показатели 4-го уровня. Если атрибут качества определяется всего одной метрикой, то уровень метрики опускается. Аналогично может быть опущен уровень оценочного элемента, если он один определяет заданное метрикой свойство. Число оценочных элементов, входящих в метрику, не ограничивается.

Для количественного выражения характеристик качества на всех уровнях (показатели, атрибуты, метрики, оценочные элементы) принимается единая шкала оценки от 0 до 1.

4.2 Качество и жизненный цикл программных систем

Процесс управления качеством на каждой стадии жизненного цикла программного средства предполагает необходимость рассмотрения разнообразные аспекты качества и своевременное изменение представлений о качестве в ходе процессов системного анализа, проектирования, производства и эксплуатации. Стандарт ГОСТ Р ИСО/МЭК 9126-93 основан на варьировании взглядов на качество продукта в зависимости от стадии жизненного цикла. Необходимо так же учитывать, что каждая из характеристик качества имеет разный уровень значимости при формировании единой точку зрения на качество ПС.

Сокращение жизненного цикла ПС и повышение требований к их качеству является наиболее характерными особенностями развития сферы информатизации на современном этапе [8]. Следует указать, что накопленный в программной инженерии положительный опыт нашел отражение в целом ряде международных стандартов, содержащих требования к жизненному циклу автоматизированных систем обработки информации и управления, их ПО и процессов их разработки.

Так как для любого продукта, потребительского, инженерного,

программного, существует множество интерпретаций качества, то и характеристики качества могут требоваться в различной степени, могут отсутствовать или задавать определенные требования, определяя тем самым определенный компромисс между различными сторонами качества. Стоимость процесса анализа качества ПО также может быть дифференцирована на стоимость предупреждения дефектов, стоимость оценки показателей, стоимость устранения последствий внутренних и внешних сбоев.

Международные стандарты определяют две связанные модели качества программного обеспечения (ГОСТ Р ИСО/МЭК 9126-93) – внутреннее качество, внешнее качество, а также набор необходимых работ по оценке качества ПО. Качество программного кода и системной архитектуры, управление качеством и качество программной инженерии имеют прямое отношение к качеству создаваемого программного продукта [2].

Качество способно повышаться за счет итеративности циклического процесса постоянного улучшения ПО, что требует четкого контроля, координации и обратной связи в процессе управления несколькими одновременно выполняемыми процессами по обеспечению качества: обнаружения, устранения и предотвращения сбоев/дефектов и процессов улучшения качества. Для реализации требований предъявляемых международными стандартами, регламентирующими жизненный цикл программного средства существует ряд наиболее эффективных методов, моделей и концепций таких как всеобщий менеджмент качества TQM, управление качеством, подтверждение качества, методология совершенствования процессов в организациях CMMI (*Capability Maturity Model Integration*), документирование и другие. Их правильное применение и способствует повышению качества деятельности организации и как следствие конкурентоспособности выпускаемой продукции.

Управление качеством ПО применяется ко всем аспектам процессов жизненного цикла ПС, артефактов проектирования и используемых ресурсов [2]. Оно определяет сами процессы: их владельцев, требования к ним, измерения характеристик качества и анализ их результатов, а также каналы обратной связи, обеспечивающие оперативное изменение качественного состояния автоматизированной системы обработки информации и управления, составной частью которого является ПО. Планирование качества ПО включает определение свойств требуемого продукта в терминах характеристик качества. Некоторые из специализированных процессов управления качеством определены в стандарте ГОСТ Р ИСО/МЭК 12207-2010. «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств», который определяет общую структуру процессов жизненного цикла ПС, в том числе и процессы обеспечения качества. К ним относятся:

- процесс менеджмента качества из группы организационного обеспечения;
- процессы оценки проекта, измерений и управления из группы процессов проектирования;
- процессы анализа системных требований и квалификационного тестирования из группы технических процессов;
- процессы обеспечения гарантий качества, верификации и аудита, из группы поддержки ПС.

Данные процессы жизненного цикла ПС служат для подтверждения того, что ПС и процессы жизненного цикла проекта соответствуют заданным требованиям обеспечения качества. Деятельность по управлению качеством проводится на основе планирования, постановки работ и исполнения набора действий, направленных на то, чтобы качество стало фундаментальным свойством ПО. Процессы обеспечения качества позволяют идентифицировать проблемы еще на ранних стадиях, что повышает эффективность любой сложной деятельности.

Подтверждение соответствия ПС предъявляемым к ней требованиям обеспечивает процесс *верификации*, который напрямую адресуется вопросам качества ПО и использует соответствующие механизмы тестирования для обнаружения дефектов и ошибок. Верификация – это деятельность, направленная на правильную разработку продукта и обеспечение создания ПС с точки зрения достижения поставленной цели. Процессы верификации, а также другие процессы, связанные с обеспечением качества ПО начинаются на ранних стадиях разработки и сопровождения.

Ключевой задачей процессов управления качеством является обеспечение процессов нахождения и классификации дефектов. Описание характеристик дефектов позволяет повысить степень понимания ПС, облегчает коррекцию процесса проектирования, информирует заказчиков ,руководителей и менеджеров проектов о состоянии процесса или программного артефакта. Управление качеством должно обеспечивать сбор данных о состоянии качества на всех стадиях разработки и сопровождения ПО [3]. По результатам работ, направленных на обнаружение дефектов, должны выполняться мероприятия по устранению дефектов из программного продукта, а также анализ и подведение итогов по обнаруженным несоответствиям, дефектам и ошибкам, использование технологий улучшения продукта и процесса, отслеживание, замена и удаление из системы дефектных элементов.

Процессы *подтверждения качества* исследуют и оценивают выходной программный продукт, связанный со спецификацией требований к ПО на предмет трассируемости, согласованности, полноты и завершенности, корректности и непосредственного выполнения требований.

Модели качества программных продуктов включают метрики для определения уровня каждой характеристики качества, присущей продукту. Совокупность используемых метрик могут способствовать поиску проблемных аспектов и узких мест в процессах управления качеством.

Из вышесказанного следует, что обеспечением качества ПС необходимо заниматься на всем протяжении ее жизненного цикла. Причем наиболее важной является начальные концептуальные стадии. Накопленный опыт в области управления качеством ПО заключен во множестве современных международных стандартов, который незаменим при производстве высококачественной конкурентоспособной продукции в сфере информационных технологий – программных средств и систем.

4.3 Тестирование ПО

Тестирование кода позволяет на протяжении всего жизненного цикла программной системы гарантировать соответствие всех атрибутов программных проектов заданным параметрам качества [8, 16, 17]. Основная цель тестирования состоит в определении отклонений при реализации функциональных требований, степени их значимости, обнаружении ошибок в ходе выполнения программ и своевременное их исправление.

На протяжении всего процесса разработки ПО необходимо применять комплексные методы проверки качества кода, включающие различные типы тестирования. При этом нужно гарантировать соответствие заданным показателям качества всех промежуточных версий проектируемой системы. Применяются как автоматические, так и ручные тесты. Основные видами тестирования ПО являются следующие.

1) *Модульное тестирование* (тестирование программных модулей) применяется для проверки правильности реализации функций, подпрограмм и методов классов ПО. Модульные тесты создаются и реализуются проектировщиками непосредственно в процессе написания кода. Как правило, модульное тестирование используется как для проверки самого кода приложения, так и для проверки функционирования информационных баз ПС.

2) *Исследовательское тестирование* предназначается для проверки качества кода в случае, когда тестировщик пытается интуитивно изучить возможности ПС, обнаружить и зафиксировать неизвестные ошибки, не имея заранее определенных сценариев тестирования.

3) *Интеграционное тестирование* применяется для проверки правильности совместного взаимодействия компонентов программного продукта.

4) *Функциональное тестирование* предназначено для проверки конкретных функциональных требований к ПО и осуществляется в основном при добавлении в систему новых функций.

5) *Нагрузочное тестирование* применяется для проверки работоспособности ПС при максимальной (предельной) входной нагрузке.

6) *Регрессионное тестирование* применяется в случае внесения изменений в ПО для того, чтобы проверить корректность работы тех компонентов системы, которые могут взаимодействовать с измененным компонентом.

7) *Комплексное тестирование* осуществляется для проверки полноты всех функциональных и нефункциональных требований ПС.

8) *Приемочное тестирование* является завершающим функциональным испытанием, которое должны подтвердить, что созданная программная система соответствует требованиям и пожеланиям пользователей и заказчиков. Приемочные тесты создаются независимыми специалистами по контролю качества ПО и тестировщиками.

Современные инструментальные среды разработки приложений, такие как *Microsoft VisualStudio* последних версий, предоставляют разработчикам программного обеспечения возможности создавать модульные и нагрузочные тесты, а также тесты для проверки пользовательского интерфейса.

Для этого используются шаблоны тестовых проектов, к которым можно отнести следующие:

- макет модульного теста, который позволяет создавать тесты проверки модулей в процессе проектирования;
- макет с Web –ориентированными тестами для оценки производительности и нагрузочными тестами;
- макет с кодированными тестами пользовательского интерфейса.

Кроме того, в среду разработки вводится инструментарий тестировщика, который предназначен для управления процессом тестирования ПО, включая планирование, тестирование, мониторинг и контроль.

Примером такого инструментария является подсистема *Microsoft Test Manager* в среде разработки в *Microsoft VisualStudio*. С помощью таких средств специалисты по тестированию могут разрабатывать планы и управляют процессом тестированием.

При формировании плана тестирования в него можно добавлять как наборы тестов и тестовые ситуации, так и конфигурации, которые необходимы для проведения тестирования. Конфигурации применяются для определения среды, в которой будут реализовываться сформированные наборы тестов. При этом, как правило, имеется возможность выполнять тестирование как в ручном, так и автоматическом режимах. Результаты тестирования сохраняются в базе данных, что позволяет их анализировать и создавать различные отчеты. Ошибки, выявляемые в процессе тестирования, регистрируются, документируются и передаются разработчикам для устранения.

Одним из наиболее эффективных методов улучшения качества кода программных приложений является *рефакторинг*, который представляет собой процесс изменения программной системы так, чтобы её внешнее поведение не изменялось, а улучшалась внутренняя структура. Это означает, что для создания качественного программного продукта в процессе проектирования необходимо обеспечивать не только выполнение функциональных требований, но и нефункциональных, таких как удобство сопровождения, предполагающее возможность и простоту внесения изменений в код программы, а также обеспечение легкости его понимания.

При тестировании программного кода и его оформления, следует проверять те основные признаки, которые определяют снижение качества программы. К ним относятся:

- *жесткость*, характеристику программы, определяющую степень сложности внесения изменений в код;
- *хрупкость*, свойство ПО повреждаться одновременно в нескольких местах при внесении единственного изменения;
- *ненужная сложность*, свойство, которое характеризует факт наличия в программе элементов, которые в текущий момент не используются;
- *косность*, характеристику того, что в коде содержатся структуры или фрагменты, которые были бы полезными в других ПС, но трудоемкость операций и риски, сопряженные с попыткой отделить эти части от оригинальной системы, слишком велики;
- *ненужные повторения*, характеристику, показывающую наличие повторяющихся фрагментов кода;
- *непрозрачность*, свойство, оценивающее трудность кода для понимания.

4.4 Методика комплексной оценки качества ПС

Характеристики качества в типовой методике оценки качества ПС представляются иерархической структурой, в которую включены комплексные и единичные характеристики (метрики и оценочные элементы). Как уже отмечалось, на всех уровнях показателей для метрик характеристик качества принята единая шкала оценки от 0,1 до 1, где 1 соответствует высшему качеству. Для рейтинговых оценок, при сопоставлении нескольких программных продуктов, может использоваться другая шкала - от 0,1 до 10.

Метрики качества для единичных показателей определяют в зависимости от метода измерения (экспертной оценки). Метрики комплексных показателей определяют на основе связанных с ними показателей нижестоящего уровня с использованием вычислительных методов или специально разработанных правил. Для комплексных характеристик качества вычисляются

средневзвешенные значения, для чего для каждого слагаемого значения характеристики более низкого уровня должны быть установлены весовые коэффициенты, определяющие значимость этой характеристики. Сумма всех весовых коэффициентов должна быть равна единице.

Правила принятия решений о подтверждении качестве ПО определяют критерии, выработанные на основе требований и результатов измерения (экспертной оценки) показателей качества. Требования к качеству задаются базовыми (эталонными) значениями. В процессе комплексной оценки качества ПС на каждом уровне, не считая уровни единичных характеристик, оцениваемых непосредственно, проводятся вычисления показателей качества ПС. При этом определяются абсолютные количественные значения показателей P_{ij}, где j – порядковый номер характеристики данного уровня для i-го показателя вышестоящего уровня. При этом каждая характеристика качества 1-го, 2-го и 3-го уровней (показатели, атрибуты и метрики) определяется двумя числовыми параметрами – количественным значением оценки и весовыми коэффициентами V_{ij}. Под нулевым уровнем будем считать совокупное качество всей ПС.

Сумма весовых коэффициентов показателей уровня V_{ij}, относящихся к j −ой характеристике вышестоящего уровня $(i-1)$, есть величина постоянная и принимается равной 1:

$$\sum_{j=1}^{n} V_{ij} = 1, \qquad (33)$$

где j – порядковый номер характеристики, n – число характеристик данного уровня, относящихся к i-ой характеристике вышестоящего уровня.

Комплексная оценка качества ПС в целом формируется группой экспертов по фактическому набору оценок показателей качества. Для оценки качества ПС различного назначения методом экспертного опроса необходимо составить таблицу значений базовых показателей качества ПС. Определение усредненной оценки характеристики по нескольким его значениям его подхарактеристик проводится по формуле:

$$P_{ki} = \sum_{j=1}^{n} V_{ij} P_{ij}, \qquad (34)$$

где $k = \begin{cases} i-1, i>1 \\ 0, i=1 \end{cases}$.

Если уровень характеристики не является последним, то есть общей комплексной оценкой качества ПС в целом, то процесс подсчета характеристик в порядке повышения уровня продолжается итеративно.

Общая оценка качества ПС осуществляется путем сравнения полученных расчетных значений показателей с соответствующими значениями существующего аналога или образцовой ПС, принимаемой за эталонный образец.

Значения показателей ПС должны также соответствовать значениям, отражающим современный уровень качества и прогнозируемый мировой уровень. В качестве этих систем для сравнения выбираются реально существующие ПС того же функционального назначения, с сопоставимыми основными параметрами, имеющими подобную структуру и применяемые в сходных условиях эксплуатации.

При выполнении практических работ по оценке качества программных систем необходимо выполнить следующее.

1) Выбрать показатели и атрибуты качества в соответствии с ГОСТ Р ИСО/МЭК 9126-93 и сформулировать их сущность. Структура показателей и атрибутов в общем случае быть как полной, так и ограниченной в зависимости от назначения о областей применения ПС. При выборе структуры показателей и атрибутов можно использовать рекомендации, содержащиеся в ГОСТ 28195-89. При этом каждый показатель или атрибут должен быть существенным, т. е. должны быть ясны потенциальные выгоды его использования. Показатели представляются в виде таблицы.

2) Установить весовые коэффициенты показателей и атрибутов с учетом правила (33).

3) Для каждого атрибута (характеристики 2-го уровня) выбрать одну или несколько метрик (характеристик 3-го уровня), при выборе можно руководствоваться рекомендательным перечнем метрик, приведенным в ГОСТ 28195-89.

4) Установить весовые коэффициенты для каждой метрики с учетом правила (33).

5) В случае если значение метрики не было определено с помощью оценочных элементов (см. п. 6), тогда для каждой метрики определить путем измерения или экспертного заключения количественное значение от 0 до 1, исходя из следующего: 0 - качество по данной метрике является неприемлемым, а при 1 отмечен очень высокий уровень качества. Можно присваивать промежуточные значения в соответствии с мнением эксперта относительно полезности проверяемого свойства ПС.

6) При необходимости декомпозиции метрики выбрать несколько оценочных элементов (характеристик 4-го уровня), установить их весовые коэффициенты, дать количественную оценку элементу и определить оценку метрики по правилу (34).

7) Определить вычислением по правилу (34) взвешенную сумму каждого атрибута качества, использую оценки метрик, определяющих этот атрибут.

8) Определить вычислением по правилу (34) взвешенную сумму каждого показателя качества, использую оценки атрибутов, определяющих этот показатель в соответствии ГОСТ Р ИСО/МЭК 9126-93.

9) Определить совокупное качество ПС как взвешенную сумму отдельных показателей по правилу (34).

10) Представить выходные данные в виде сводной таблицы, пример которой представлен в табл. 2.

11) Сформулировать заключение по проведенной экспертизе качества ПС, в котором:

- дать обобщенную оценку качества ПС с точки зрения соответствия требованиям, изложенным в техническом задании или в спецификации качества;
- выделить проблемные места в иерархической системе характеристик качества, сформулировать возможные причины их появления;
- привести рекомендации по целесообразности и возможности устранения выявленных проблем в качестве ПС;
- дать общую оценку сложности и трудоемкости проведенной экспертизы качества ПС.

Таблица 2 – Пример комплексной оценки программной системы

№	Атрибут качества	Сущность атрибутов	Метрика (оценочный элемент)	Вес	Оценка
Показатель «Надежность»					
1	Стабильность	Частота отказов при ошибках в ПО.	1. Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных	0,6	0,5
			2. Наличие требований к программе по восстановлению результатов при отказах процессора, ОС	0,4	0,1
			Итого по атрибуту	0.3	0,34
2	Устойчивость к дефектам	Способность поддерживать определенный уровень качества	1. Возможность обработки ошибочных ситуаций	0,7	0,6

		функционирования в случаях программных ошибок или нарушения определенного интерфейса.	2. Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления	0,3	0,8
			Итого по атрибуту	0.4	0.38
3	Восстанавливаемость	Возможность восстанавливать уровень качества функционирования и восстанавливать данные, непосредственно поврежденные в случае отказа, а также к времени и усилиям, необходимым для этого.	1. Наличие средств, обеспечивающих завершение процесса решения в случае помех	0,7	0,5
			2. Наличие средств, обеспечивающих выполнение программы в сокращенном объеме в случае ошибок	0,3	0,5
			Итого по атрибуту	0.3	0.5
			Итого по показателю	0.2	0.40
Показатель «Сопровождаемость»					
1	Анализируемость	Способность к диагностике недостатков или случаев отказов, или определения составных частей для модернизации	1. Простота архитектуры проекта	0,6	0,8
			2. Использование типовых компонентов ПС	0,4	0,7
			Итого по атрибуту	0.2	0,76
2	Изменяемость	Возможность модификации, устранению отказа или для изменения условий эксплуатации.	1. Простота архитектуры проекта	0,6	0,8
			2. Межмодульные связи	0,4	0,6
			Итого по атрибуту	0.2	0,72
3	Стабильность	Частота отказов при ошибках в ПО	Соблюдения правила нисходящего программирования	1	0,6
			Итого по атрибуту	0.3	0.6
4	Тестируемость	Усилия, необходимые для проверки модифици-	1.Простота кодирования	0,8	0,8

		рованного ПО.	2.Комментарии логики программ	0,2	0,5
			Итого по атрибуту	0.3	0.74
			Итого по показателю	0.1	0.70
Показатель «Практичность»					
1	Понятность	Понимание общей логической концепции и ее применимости.	1. Полнота пользовательской документации	0,4	0,7
			2. Понятность пользовательской документации	0,6	0,6
			Итого по атрибуту	0.4	0.64
2	Простота использования	Усилия пользователя по эксплуатации и оперативному управлению.	1. Удобство эксплуатации	0,5	0,8
			2. Управление меню	0,5	0,8
			Итого по атрибуту	0.3	0.8
3	Обучаемость	Усилия пользователя по обучению его применению (например, оперативному управлению, вводу, выводу).	1. Документация для освоения	0,4	0,9
			2. Легкость освоения	0,6	0,7
			Итого по атрибуту	0.3	0.78
			Итого по показателю	0.2	0.73
Показатель «Эффективность»					
1	Временная эффективность	Времена отклика и обработки и к скоростям выполнения его функций.	1.Скорость обработки информации	0,6	0,8
			2.Быстрота отклика системы	0,4	0,8
			Итого по атрибуту 0,5	0.7	0.8
2	Использование ресурсов	Объем используемых ресурсов и продолжительности такого использования при выполнении функции.	1.Использование оперативной памяти	0,6	0.9
			2.Занимаемость ЦП	0,4	0.9
			Итого по атрибуту	0.3	0.9
			Итого по показателю	0.1	0,83
Показатель «Функциональность»					

1	Корректность	Способность выполнять точные задачи так, как они определены их спецификацией	1. Правильное оформление всех частей документов	0,2	0,5
			2. Наличие описания всех параметров	0,2	0,5
			3. Реализация всех основных функций	0,6	0,8
			Итого по атрибуту	0.3	0,68
2	Пригодность для применения	Наличие и соответствию набора функций конкретным задачам.	1. Требуемый объем внутренней памяти	0,7	0,9
			2. Требуемое базовое программное обеспечение	0,3	0,9
			Итого по атрибуту	0.2	0.9
3	Способность к взаимодействию	Способность взаимодействовать с конкретными системами.	1. Единообразие способов восстановления информации для возврата	0,2	0,1
			2. Единообразие способов сохранения информации для возврата	0,8	0,9
			Итого по атрибуту	0.1	0,74
4	Защищенность	Способность предотвращать несанкционированный доступ, случайный или преднамеренный, к программам и данным.	1. Наличие средств защиты (например, шифрование тестов)	0,8	0,1
			2. Возможность работы локальной компьютерной сети	0,2	0,3
			Итого по атрибуту	0.4	0.14
			Итого по показателю	0.2	0.51
Показатель «Мобильность»					
1	Адаптируемость	Степень адаптации к различным конкретным условиям эксплуатации, без применения других действий или способов,	1. Широта охвата функций	0,6	0,8
			2. Простота архитектуры проекта	0,4	0,8
			Итого по атрибу-	0.3	0.8

		кроме тех, что предназначены для этого в рассматриваемом ПО.	*му*		
2	Простота установки	Усилия, необходимые для внедрения ПО в конкретное окружение.	Легкость установки ПС	1	1
			Итого по атрибуту	0.2	1
3	Сосуществование	Степень подчиненности стандартам или соглашениям, относящимся к мобильности.	1.Зависимость от базового ПО	0,5	0,4
			2.Зависимость от используемого комплекта технических средств	0,5	0,6
			Итого по атрибуту	0.2	0.5
4	Взаимозаменяемость	Простота и трудоемкости его применения вместо другого конкретного программного средства в среде этого средства.	Возможность использовать ПС вместо другого ПО	1	0,9
			Итого по атрибуту	0.3	0.9
			Итого по показателю	0.2	0,81
ИТОГО ПК = 0,2*0,40+0,1*0,70+0,3*0,78+0,3*0,9+0,2*0,51+0,2*0,81=0.6444					

4.5 Контрольные вопросы и задания

1) Дате определение жизненного цикла продукции.

2) Какие процессы жизненного цикла программной системы тесно связаны с обеспечением их качества?

3) Каковы основные цели имеет тестирование программного средства? Какие задачи при этом возникают?

4) Каким образом осуществляется оценка технико-экономических показателей разработки программных средств?

5) Какие уровни тестирования программных систем используются в процессе их проектирования?

6) Разработайте сценарий тестирования для одного или нескольких следующих модулей:

- решение квадратного уравнения.
- определение площади треугольника.
- интегрирование функции.
- редактор строки.
- суммирование ряда вещественных чисел.

7) Каким образом осуществляется количественная оценка показателей качества программных средств?

8) Каким образом осуществляется процесс подтверждения качества программной системы? Что для этого необходимо?

9) На примере конкретной программной системы выполните экспертизу ее качества согласно приведенной методике.

ЗАКЛЮЧЕНИЕ

В учебном пособии систематизированы сведения, связанные с созданием качественных и надежных программных систем в контексте современных тенденций в области управления качеством продукции широкого применения. Отмечены системные свойства деятельности по управлению качеством, которая рассматривается как масштабная и разноплановая совокупность процессов, направленных не только на обеспечение качества конкретных программных продуктов, но и на повышение эффективности предприятий и организаций, применяющих их.

Важное значение в области управления качеством ПО имеет международная стандартизация, позволяющая огромному количеству разработчиков и пользователей ПО использовать единые методологические подходы по оценке и обеспечению качества программных продуктов. Изучение ключевых стандартов, регламентирующих систему показателей и методов обеспечения качества ПО, должно входить в систему подготовки специалистов в области информационных технологий.

Рассмотрены вопросы надежности программно-аппаратных комплексов, при этом сделан краткий обзор теории надежности технических систем, указаны основные отличия ПО с точки зрения надежности, определяющие существенные отличия подходов к исследованию и обеспечению надежности программных и аппаратных средств современных вычислительных систем. Изложены основные принципы разработки надежного ПО.

Особое внимание в пособии уделено методическим основам обеспечения ключевых показателей качества программных систем: функциональности, надежности, сопровождаемости, удобства применения, мобильности и эффективности, при этом даны практические рекомендации по обеспечению высокого качества этих показателей.

Рассмотрены вопросы измерений и оценки качества программных систем с использованием иерархической системы характеристик качества. Приведена типовая методика экспертизы качества ПС, которая может быть использована студентами при решении практических задач, связанных с анализом качества программных продуктов различного назначения.

Для контроля и закрепления знаний, полученных студентами при изучении данного пособия, каждая глава имеет перечень контрольных вопросов и заданий. Приведенный библиографический перечень позволит им осуществить более глубокое изучение отдельных тем и вопросов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Липаев В.В. Сертификация программных средств. – М.: СИНТЕГ, 2010. – 348 с.

2. Перемитина Т.О. Управление качеством программных систем : учебное пособие. – Томск: Эль Контент, 2011. – 228 с.

3. Огвоздин В.Ю. Управление качеством. Основы теории и практики. – М.: Дело и Сервис, 2009. – 304 с.

4. Лапидус В.А. Всеобщее качество (TQM). – М.: Новости, 2000. – 432 с.

5. Международный стандарт ISO 9000:2005 «Системы менеджмента качества. Основные положения и словарь». – М.: Стандартинформ, 2006. – 37 с.

6. ГОСТ 28195-89. Оценка качества программных средств. Общие положения. – М.: Изд-во стандартов, 2001. – 32 с.

7. ГОСТ Р ИСО/МЭК 9126-93. Оценка программной продукции. Характеристики качества и руководства по их применению. – М.: Изд-во стандартов, 2004. – 9 с.

8. Белик А.Г., Цыганенко В.Н. Проектирование и архитектура программных систем: учебное пособие. – Омск : Изд – во ОмГТУ, 2016. – 80 с.

9. Холстед М.Х. Начало науки о программах. – М.: Финансы и Статистика, 1981. - 128 с.

10. Воскобоев В.Ф. Надёжность технических систем и техногенный риск. Ч. I. Надёжность технических систем. – М.: ООО ИД «Альянс», 2008.– 200 с.

11. Ямпурин Н.П., Баранова А.В. Основы надежности электронных средств: учебное пособие. – М.: Академия, 2010. – 242 с.

12. Майерс Г. Надежность программного обеспечения. – М.: Мир. – 360 с.

13. Черкесов Г.Н. Надежность аппаратно-программных комплексов: учебное пособие. – СПб.: Питер, 2005. – 478 с.

14. Андон Ф.И. и др. Основы инженерии качества программных систем. – Киев: Академпериодика, 2007. – 680 с.

15. Соммервил И. Инженерия программного обеспечения. – М.: Вильямс, 2002. – 623 с.

16. Константайн Л., Локвуд Л. Разработка программного обеспечения. – СПб.: Питер, 2004. – 592с.

17. Криспин Л., Грегори Д. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд. – М.: Вильямс, 2010. – 464 с.

Учебное издание

Белик Алевтина Георгиевна
Цыганенко Валерий Николаевич

КАЧЕСТВО И НАДЕЖНОСТЬ ПРОГРАММНЫХ СИСТЕМ

Учебное пособие

Редактор *О. В. Маер*
Компьютерная верстка *О. Н. Савостеевой*

Сводный темплан 2018 г.
Подписано в печать 30.01.2018 г. Формат 60×84 $^{1}/_{16}$. Бумага офсетная.
Отпечатано на дупликаторе. Уч. изд. – л. 5,0. Усл. – печ. л. 5,0.
Тираж 50 экз. Заказ XX.

Издательство ОмГТУ. 644050, г. Омск, пр. Мира, 11; т. 23 – 02 – 12
Типография ОмГТУ