Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения
информационных технологий

С. С. Куликов, Г. В. Данилова

ТЕСТИРОВАНИЕ ВЕБ-ОРИЕНТИРОВАННЫХ ПРИЛОЖЕНИЙ

*Рекомендовано УМО по образованию в области информатики
и радиоэлектроники для специальности
1-40 01 01 «Программное обеспечение
информационных технологий»
в качестве учебно-методического пособия*

Минск БГУИР 2017

УДК 004.414.3(076.5)
ББК 32.973.26-018.2я73
К90

Р е ц е н з е н т ы:

кафедра информатики и веб-дизайна
учреждения образования
«Белорусский государственный технологический университет»
(протокол №3 от 13.10.2016);

доцент кафедры
многопроцессорных систем и сетей
Белорусского государственного университета,
кандидат физико-математических наук, доцент Т. В. Соболева

Куликов, С. С.

К90 Тестирование веб-ориентированных приложений : учеб.-метод. пособие / С. С. Куликов, Г. В. Данилова. – Минск : БГУИР, 2017. – 100 с. : ил.

ISBN 978-985-543-328-7.

Содержит материалы к практическим занятиям, включающие вопросы и задания.

УДК 004.414.3(076.5)
ББК 32.973.26-018.2я73

ISBN 978-985-543-328-7

СОДЕРЖАНИЕ

1 ТЕСТИРОВАНИЕ ДОКУМЕНТАЦИИ И ТРЕБОВАНИЙ

1.1 Что такое «требование»

Всё так или иначе начинается с документации и требований.

!!!	**Требование** (requirement[1]) – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

....	Небольшое «историческое отступление»: если поискать определения требования в литературе 10-, 20-, 30-летней давности, то можно заметить, что изначально о пользователях, их задачах и полезных для них свойствах приложения в определении требования не было сказано. Пользователь выступал некоей абстрактной фигурой, не имеющей отношения к приложению. В настоящее время такой подход недопустим, т. к. он не только приводит к коммерческому провалу продукта на рынке, но и многократно повышает затраты на разработку и тестирование.

....	Хорошим кратким предисловием ко всему тому, что будет рассмотрено в данной главе, можно считать небольшую статью «What is documentation testing in software testing»[2].

1.2 Важность требований

Требования являются отправной точкой для определения того, что проектная команда будет проектировать, реализовывать и тестировать. Элементарная логика говорит нам, что если в требованиях что-то «не то», то и реализовано будет «не то», т. е. колоссальная работа множества людей будет выполнена впустую. Эту мысль иллюстрирует рисунок 1.а.

Брайан Хэнкс, описывая важность требований[3], подчёркивает, что они:

- Позволяют понять, что и с соблюдением каких условий система должна делать.
- Предоставляют возможность оценить масштаб изменений и управлять изменениями.

[1] **Requirement.** A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. ISQTQB glossary.

[2] What is documentation testing in software testing. URL: http://istqbexamcertification.com/what-is-documentation-testing/.

[3] Requirements in the Real World / B. Hanks URL: https://classes.soe.ucsc.edu/cmps109/Winter02/notes/requirementsLecture.pdf.

- Являются основой для формирования плана проекта (в том числе плана тестирования).
- Помогают предотвращать или разрешать конфликтные ситуации.
- Упрощают расстановку приоритетов в наборе задач.
- Позволяют объективно оценить степень прогресса в разработке проекта.

Вне зависимости от того, какая модель разработки ПО используется на проекте, чем позже будет обнаружена проблема, тем сложнее и дороже будет её решение. А в самом начале («водопада», «спуска по букве v», «итерации», «витка спирали») идёт планирование и работа с требованиями.

Если проблема в требованиях будет выяснена на этой стадии, её решение может свестись к исправлению пары слов в тексте, в то время как недоработка, вызванная пропущенной проблемой в требованиях и обнаруженная на стадии эксплуатации, может даже полностью уничтожить проект.

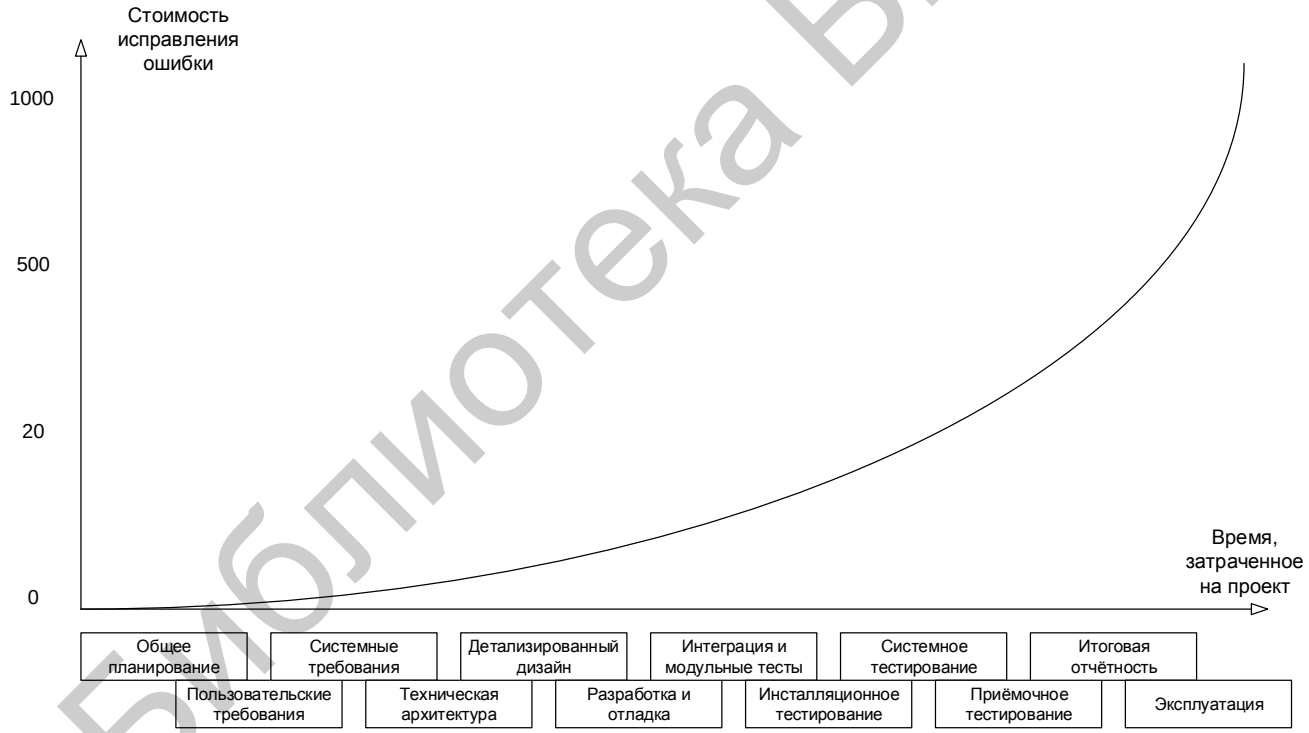


Рисунок 1.а – Стоимость исправления ошибки в зависимости от момента её обнаружения

Если графики вас не убеждают, попробуем проиллюстрировать ту же мысль на простом примере. Допустим, вы с друзьями составляете список покупок перед поездкой в гипермаркет. Вы поедете покупать, а друзья ждут вас дома. Сколько «стоит» дописать, вычеркнуть или изменить пару пунктов, пока вы только-только составляете список? Нисколь-

ко. Если мысль о несовершенстве списка настигла вас по пути в гипермаркет, уже придётся звонить (дёшево, но не бесплатно). Если вы поняли, что в списке «что-то не то» в очереди на кассу, придётся возвращаться в торговый зал и тратить время. Если проблема выяснилась по пути домой или даже дома, придётся вернуться в гипермаркет. И, наконец, «клинический» случай: в списке изначально было что-то уж совсем неправильное (например, «100 кг конфет – и всё»), поездка совершена, все деньги потрачены, конфеты привезены и только тут выясняется, что «ну, мы же пошутили».

Задание 1.a: представьте, что ваш с друзьями бюджет ограничен, и в списке требований появляются приоритеты (что-то купить надо обязательно, что-то, если останутся деньги, и т. п.). Как это повлияет на риски, связанные с ошибками в списке?

Ещё одним аргументом в пользу тестирования требований является то, что по разным оценкам в них зарождается от ½ до ¾ всех проблем с программным обеспечением. В итоге есть риск, что получится так, как показано на рисунке 1.b.

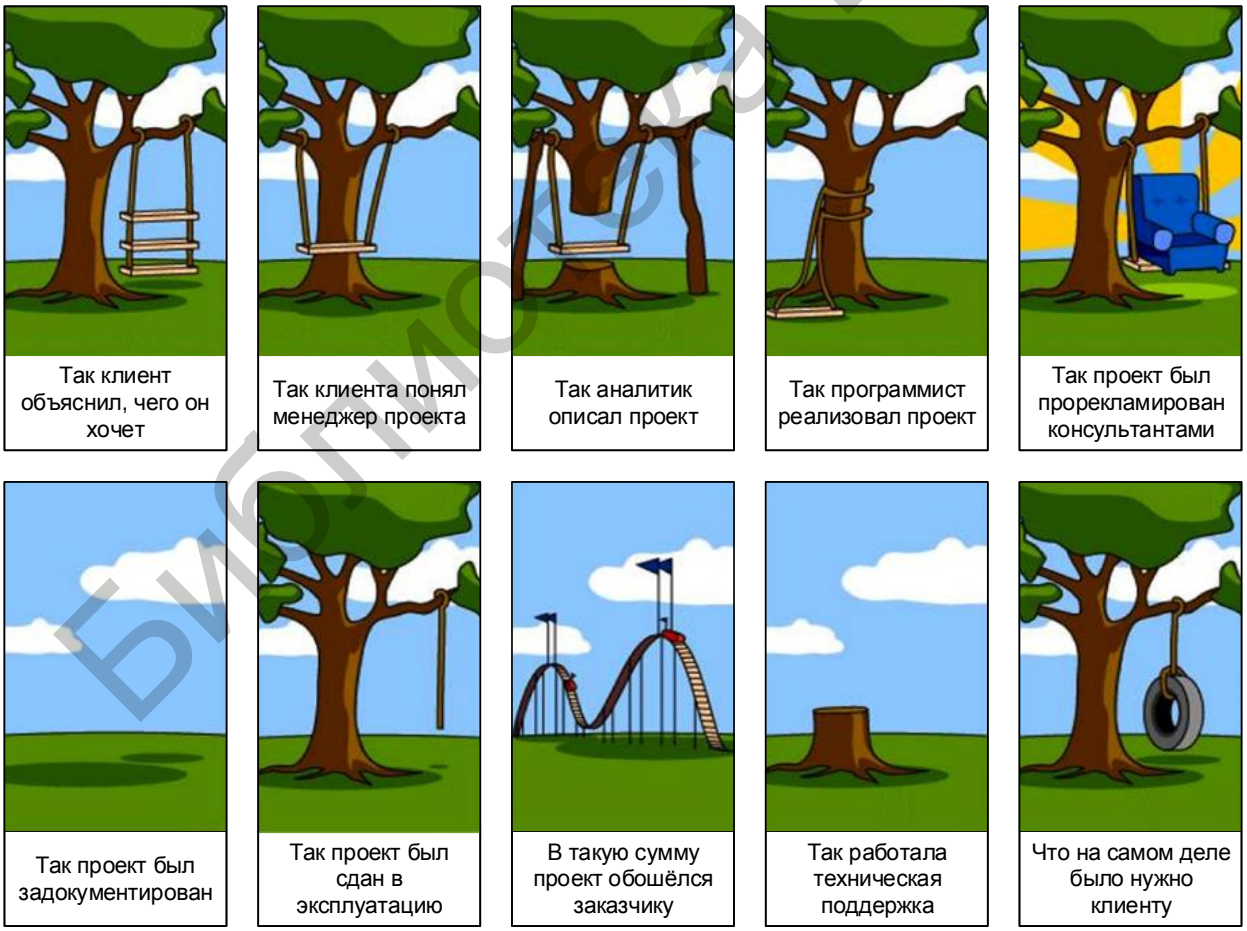


Рисунок 1.b – Типичный проект с плохими требованиями

Поскольку мы постоянно говорим «документация и требования», а не просто «требования», то стоит рассмотреть перечень документации, которая должна подвергаться тестированию в процессе разработки ПО (хотя далее мы будем концентрироваться именно на требованиях).

В общем случае документацию можно разделить на два больших вида в зависимости от времени и места её использования (здесь будет очень много сносок с определениями, т. к. по видам документации очень часто поступает множество вопросов и потому придётся рассмотреть некоторые моменты подробнее).

- **Продуктная документация** (product documentation, development documentation[4]) используется проектной командой во время разработки и поддержки продукта. Она включает:
  - План проекта (project management plan[5]) и в том числе тестовый план (test plan[6]).
  - Требования к программному продукту (product requirements document, PRD[7]) и функциональные спецификации (functional specifications[8] document, FSD[9]; software requirements specification, SRS[10]).

[4] Development documentation. Development documentation comprises those documents that propose, specify, plan, review, test, and implement the products of development teams in the software industry. Development documents include proposals, user or customer requirements description, test and review reports (suggesting product improvements), and self-reflective documents written by team members, analyzing the process from their perspective. Barker Thomas T. Documentation for software and IS development.

[5] Project management plan. A formal, approved document that defines how the project is execu-ted, monitored and controlled. It may be summary or detailed and may be composed of one of more subsidiary management plans and other planning documents. PMBOK. – 3rd ed.

[6] Test plan. A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process. ISTQB Glossary.

[7] Product requirements document, PRD. The PRD describes the product your company will build. It drives the efforts of the entire product team and the company's sales, marketing and customer support efforts. The purpose of the product requirements document (PRD) or product spec is to clearly and unambiguously articulate the product's purpose, features, functionality, and behavior. The product team will use this specification to actually build and test the product, so it needs to be complete enough to provide them the information they need to do their jobs. Cagan M. How to write a good PRD.

[8] Specification. A document that specifies, ideally in a complete, precise and verifiable manner, the requirements, design, behavior, or other characteristics of a component or system, and, often, the procedures for determining whether these provisions have been satisfied. ISTQB Glossary.

[9] Functional specifications document, FSD. См. «Software requirements specification, SRS».

[10] Software requirements specification, SRS. SRS describes as fully as necessary the expected behavior of the software system. The SRS is used in development, testing, quality assurance, project management, and related project functions. People call this deliverable by many different names, including business requirements document, functional spec, requirements document, and others. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

- Архитектуру и дизайн (architecture and design[11]).
- Тест-кейсы и наборы тест-кейсов (test cases[12], test suites[13]).
- Технические спецификации (technical specifications[14]), такие как схемы баз данных, описания алгоритмов, интерфейсов и т. д.

- **Проектная документация** (project documentation[15]) включает в себя как продуктную документацию, так и некоторые дополнительные виды документации и используется не только на стадии разработки, но и на более ранних и поздних стадиях (например, на стадии внедрения и эксплуатации). Она включает:
  - Пользовательскую и сопроводительную документацию (user and accompanying documentation[16]), такую как встроенная помощь, руководство по установке и использованию, лицензионные соглашения и т. д.
  - Маркетинговую документацию (market requirements document, MRD[17]), которую представители разработчика или заказчика используют как на начальных этапах (для уточнения сути и концепции проекта), так и на финальных этапах развития проекта (для продвижения продукта на рынке).

В некоторых классификациях часть документов из продуктной документации может быть перечислена в проектной документации – это

[11] **Architecture. Design.** A software *architecture* for a system is the structure or structures of the system, which comprise elements, their externally-visible behavior, and the relationships among them. … *Architecture is design*, but not all design is architecture. That is, there are many design decisions that are left unbound by the architecture, and are happily left to the discretion and good judgment of downstream designers and implementers. The architecture establishes constraints on downstream activities, and those activities must produce artifacts (finer-grained designs and code) that are compliant with the architecture, but architecture does not define an implementation. Documenting Software Architectures / P. Clements [et al.]. Очень подробное описание различия архитектуры и дизайна можно найти в статье Eden A., Kazman R. Architecture, Design, Implementation. URL: http://www.eden-study.org/articles/2003/icse03.pdf.

[12] **Test case.** A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. ISTQB Glossary.

[13] **Test suite.** A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one. ISTQB Glossary.

[14] **Technical specifications.** Scripts, source code, data definition language, etc. PMBOK, 3rd ed. Также см. «Specification».

[15] **Project documentation.** Other expectations and deliverables that are not a part of the software the team implements, but that are necessary to the successful completion of the project as a whole. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[16] **User documentation.** User documentation refers to the documentation for a product or service provided to the end users. The user documentation is designed to assist end users to use the product or service. This is often referred to as user assistance. The user documentation is a part of the overall product delivered to the customer. На основе статей doc-department.com.

[17] **Market requirements document, MRD.** An MRD goes into details about the target market segments and the issues that pertain to commercial success. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

совершенно нормально, т. к. понятие проектной документации по определению является более широким. Поскольку с этой классификацией связано очень много вопросов и непонимания, отразим суть ещё раз – графически (рисунок 1.c) – и напомним, что мы договорились классифицировать документацию по признаку того, где (для чего) она является наиболее востребованной.

Рисунок 1.c – Соотношение понятий «продуктная документация» и «проектная документация»

Степень важности и глубина тестирования того или иного вида документации и даже отдельного документа определяется большим количеством факторов, но неизменным остаётся общий принцип: всё, что мы создаём в процессе разработки проекта (даже рисунки маркером на доске, даже письма, даже переписку в скайпе), можно считать документацией и так или иначе подвергать тестированию (например, вычитывание письма перед отправкой – это тоже своего рода тестирование документации).

1.3 Источники и пути выявления требований

Требования «начинают свою жизнь» на стороне заказчика. Их сбор (gathering) и выявление (elicitation) осуществляются с помощью следующих основных техник[18] (рисунок 1.d).

[18] Здесь можно почитать подробнее о том, в чём разница между сбором и выявлением требований. Brandenburg L. Requirements Gathering vs. Elicitation. URL: http://www.bridging-the-gap.com/requirements-gathering-vs-elicitation/.

Интервью. Самый универсальный путь выявления требований, заключающийся в общении проектного специалиста (как правило, специалиста по бизнес-анализу) и представителя заказчика (или эксперта, пользователя и т. д.) Интервью может проходить в классическом понимании этого слова (беседа в виде «вопрос – ответ»), в виде переписки и т. п. Главным здесь является то, что ключевыми фигурами выступают двое – интервьюируемый и интервьюер (хотя это и не исключает наличия «аудитории слушателей», например, в виде лиц, поставленных в копию переписки).

Работа с фокусными группами. Может выступать как вариант «расширенного интервью», где источником информации является не одно лицо, а группа лиц (как правило, представляющих собой целевую аудиторию, и/или обладающих важной для проекта информацией, и/или уполномоченных принимать важные для проекта решения).

Рисунок 1.d – Основные техники сбора и выявления требований

Анкетирование. Этот вариант выявления требований вызывает много споров, т. к. при неверной реализации может привести к нулевому результату при объёмных затратах. В то же время при правильной организации анкетирование позволяет автоматически собрать и обработать огромное количество ответов от огромного количества респондентов. Ключевым фактором успеха является правильное составление анкеты, правильный выбор аудитории и правильное преподнесение анкеты.

Семинары и мозговой штурм. Семинары позволяют группе людей очень быстро обменяться информацией (и наглядно продемонстрировать те или иные идеи), а также хорошо сочетаются с интервью, анкетированием, прототипированием и моделированием – в том числе для обсуждения результатов и формирования выводов и решений. Мозговой

штурм может проводиться и как часть семинара, и как отдельный вид деятельности. Он позволяет за минимальное время сгенерировать большое количество идей, которые в дальнейшем можно не спеша рассмотреть с точки зрения их использования для развития проекта.

Наблюдение. Может выражаться как в буквальном наблюдении за некими процессами, так и во включении проектного специалиста в эти процессы в качестве участника. С одной стороны, наблюдение позволяет увидеть то, о чём (по совершенно различным соображениям) могут умолчать интервьюируемые, анкетируемые и представители фокусных групп, но с другой – отнимает очень много времени и чаще всего позволяет увидеть лишь часть процессов.

Прототипирование. Состоит в демонстрации и обсуждении промежуточных версий продукта (например, дизайн страниц сайта может быть сначала представлен в виде картинок и лишь затем свёрстан). Это один из лучших путей поиска единого понимания и уточнения требований, однако он может привести к серьёзным дополнительным затратам при отсутствии специальных инструментов (позволяющих быстро создавать прототипы) и слишком раннем применении (когда требования ещё не стабильны, и высока вероятность создания прототипа, имеющего мало общего с тем, что хотел заказчик).

Анализ документов. Хорошо работает тогда, когда эксперты в предметной области (временно) недоступны, а также в предметных областях, имеющих общепринятую устоявшуюся регламентирующую документацию. Также к этой технике относится и просто изучение документов, регламентирующих бизнес-процессы в предметной области заказчика или в конкретной организации, что позволяет приобрести необходимые для лучшего понимания сути проекта знания.

Моделирование процессов и взаимодействий. Может применяться как к «бизнес-процессам и взаимодействиям» (например: «*договор на закупку формируется отделом закупок, визируется бухгалтерией и юридическим отделом…*»), так и к «техническим процессам и взаимодействиям» (например: «*платёжное поручение генерируется модулем “Бухгалтерия”, шифруется модулем “Безопасность” и передаётся на сохранение в модуль “Хранилище”*»). Данная техника требует высокой квалификации специалиста по бизнес-анализу, т. к. сопряжена с обработкой большого объёма сложной (и часто плохо структурированной) информации.

Самостоятельное описание. Является не столько техникой выявления требований, сколько техникой их фиксации и формализации. Очень сложно (и даже нельзя!) пытаться самому «придумать требования за заказчика», но в спокойной обстановке можно самостоятельно обработать собранную информацию и аккуратно оформить её для дальнейшего обсуждения и уточнения.

Часто специалисты по бизнес-анализу приходят в свою профессию из тестирования. Если вас заинтересовало это направление, стоит ознакомиться со следующими книгами:

- Корнипаев И. Требования для программного обеспечения: рекомендации по сбору и документированию.
- Cadle J., Paul D., Turner P. Business Analysis Techniques. 72 Essential Tools for Success.
- Paul D., Yeates D., Cadle J. Business Analysis. – 2 ed.

1.4 Уровни и типы требований

Форма представления, степень детализации и перечень полезных свойств требований зависят от уровней и типов требований[19], которые схематично представлены на рисунке 1.e.

Бизнес-требования (business requirements[20]) выражают цель, ради которой разрабатывается продукт (зачем вообще он нужен, какая от него ожидается польза). Результатом выявления требований на этом уровне является общее видение (vision and scope[21]) – документ, который, как правило, представлен простым текстом и таблицами. Здесь нет детализации поведения системы и иных технических характеристик, но вполне могут быть определены приоритеты решаемых бизнес-задач, риски и т. п.

Несколько простых, изолированных от контекста и друг от друга примеров бизнес-требований:

- *Нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.*
- *Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену.*
- *Нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров.*

[19] Очень подробное описание уровней и типов требований (а также их применения) можно найти в статье Westfall L. Software Requirements Engineering: What, Why, Who, When, and How. URL: http://www.westfallteam.com/Papers/The_Why_What_Who_When_and_How_Of_Software_Requirements.pdf.

[20] **Business requirement.** Anything that describes the financial, marketplace, or other business benefit that either customers or the developing organization wish to gain from the product. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[21] **Vision and scope.** The vision and scope document collects the business requirements into a single deliverable that sets the stage for the subsequent development work. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

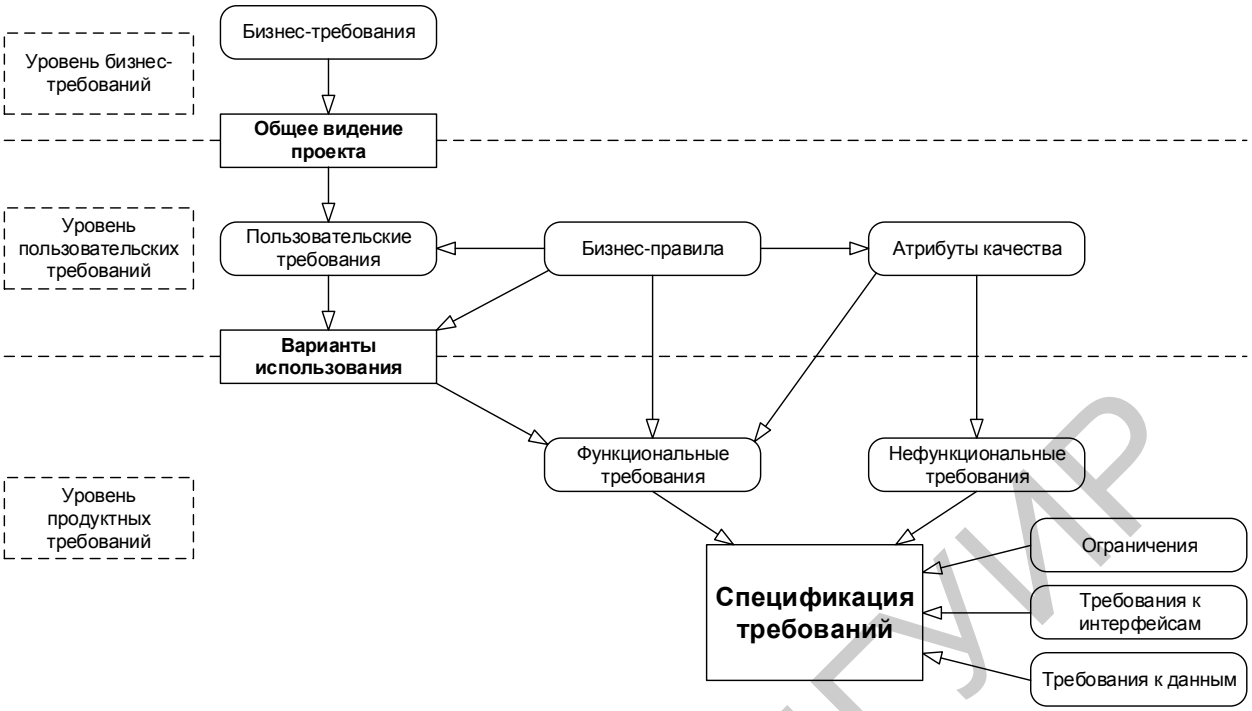


Рисунок 1.e – Уровни и типы требований

Пользовательские требования (user requirements[22]) описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы (реакцию системы на действия пользователя, сценарии работы пользователя). Поскольку здесь уже появляется описание поведения системы, требования этого уровня могут быть использованы для оценки объёма работ, стоимости проекта, времени разработки и т. д. Пользовательские требования оформляются в виде вариантов использования (use cases[23]), пользовательских историй (user stories[24]), пользовательских сценариев (user scenarios[25]). (Также см. «Создание пользовательских сценариев» в подразделе 2.5).

Несколько простых, изолированных от контекста и друг от друга примеров пользовательских требований:

- *При первом входе пользователя в систему должно отображаться лицензионное соглашение.*

[22] **User requirement.** User requirements are general statements of user goals or business tasks that users need to perform. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[23] **Use case.** A sequence of transactions in a dialogue between an actor and a component or system with a tangible result, where an actor can be a user or anything that can exchange information with the system. ISTQB Glossary.

[24] **User story.** A high-level user or business requirement commonly used in agile software development, typically consisting of one or more sentences in the everyday or business language capturing what functionality a user needs, any non-functional criteria, and also includes acceptance criteria. ISTQB Glossary.

[25] A scenario is a hypothetical story, used to help a person think through a complex problem or system. Kaner C. An Introduction to Scenario Testing. URL: http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf.

- *Администратор должен иметь возможность просматривать список всех пользователей, работающих в данный момент в системе.*
- *При первом сохранении новой статьи система должна выдавать запрос на сохранение в виде черновика или публикацию.*

Бизнес-правила (business rules[26]) описывают особенности принятых в предметной области (и/или непосредственно у заказчика) процессов, ограничений и иных правил. Эти правила могут относиться к бизнес-процессам, правилам работы сотрудников, нюансам работы ПО и т. д.

Несколько простых, изолированных от контекста и друг от друга примеров бизнес-правил:

- *Никакой документ, просмотренный посетителями сайта хотя бы один раз, не может быть отредактирован или удалён.*
- *Публикация статьи возможна только после утверждения главным редактором.*
- *Подключение к системе извне офиса запрещено в нерабочее время.*

Атрибуты качества (quality attributes[27]) расширяют собой нефункциональные требования и на уровне пользовательских требований могут быть представлены в виде описания ключевых для проекта показателей качества (свойств продукта, не связанных с функциональностью, но являющихся важными для достижения целей создания продукта – производительность, масштабируемость, восстанавливаемость). Атрибутов качества очень много[28], но для любого проекта реально важными являются лишь некоторые их подмножества.

Несколько простых, изолированных от контекста и друг от друга примеров атрибутов качества:

- *Максимальное время готовности системы к выполнению новой команды после отмены предыдущей не может превышать одну секунду.*
- *Внесённые в текст статьи изменения не должны быть утеряны при нарушении соединения между клиентом и сервером.*
- *Приложение должно поддерживать добавление произвольного количества неиероглифических языков интерфейса.*

[26] **Business rule.** A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. A business rule expresses specific constraints on the creation, updating, and removal of persistent data in an information system. Defining Business Rules – What Are They Really / D. Hay [et al.].

[27] **Quality attribute.** A feature or characteristic that affects an item's quality. ISTQB Glossary.

[28] Даже в Википедии их список огромен. URL: http://en.wikipedia.org/wiki/List_of_system_quality_attributes.

Функциональные требования (functional requirements[29]) описывают поведение системы, т. е. её действия (вычисления, преобразования, проверки, обработку и т. д.) В контексте проектирования функциональные требования в основном влияют на дизайн системы.

Несколько простых, изолированных от контекста и друг от друга примеров функциональных требований:

- *В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе.*
- *Система должна автоматически выполнять резервное копирование данных ежедневно в указанный момент времени.*
- *Электронный адрес пользователя, вводимый при регистрации, должен быть проверен на соответствие требованиям RFC822.*

Нефункциональные требования (non-functional requirements[30]) описывают свойства системы (удобство использования, безопасность, надёжность, расширяемость и т. д.), которыми она должна обладать при реализации своего поведения. Здесь приводится более техническое и детальное описание атрибутов качества. В контексте проектирования нефункциональные требования в основном влияют на архитектуру системы.

Несколько простых, изолированных от контекста и друг от друга примеров нефункциональных требований:

- *При одновременной непрерывной работе с системой 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 ч.*
- *Ни при каких условиях общий объём используемой приложением памяти не может превышать 2 ГБ.*
- *Размер шрифта для любой надписи на экране должен поддерживать настройку в диапазоне от 5 до 15 пт.*

Следующие требования в общем случае могут быть отнесены к нефункциональным, однако их часто выделяют в отдельные подгруппы (здесь для простоты рассмотрены лишь три таких подгруппы, но их может быть и гораздо больше; как правило, они проистекают из атрибутов качества, но высокая степень детализации позволяет отнести их к уровню требований к продукту).

Ограничения (limitations, constraints[31]) представляют собой факто-

[29] **Functional requirement.** A requirement that specifies a function that a component or system must perform. [ISTQB Glossary] Functional requirements describe the observable behaviors the system will exhibit under certain conditions and the actions the system will let users take. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[30] **Non-functional requirement.** A requirement that does not relate to functionality, but to attributes such as reliability, efficiency, usability, maintainability and portability. ISTQB Glossary.

[31] **Limitation, constraint.** Design and implementation constraints legitimately restrict the options available to the developer. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

ры, ограничивающие выбор способов и средств (в том числе инструментов) реализации продукта.

Несколько простых, изолированных от контекста и друг от друга примеров ограничений:

- *Все элементы интерфейса должны отображаться без прокрутки при разрешениях экрана от 800x600 до 1920x1080.*
- *Не допускается использование Flash при реализации клиентской части приложения.*
- *Приложение должно сохранять способность реализовывать функции с уровнем важности «критический» при отсутствии у клиента поддержки JavaScript.*

Требования к интерфейсам (external interfaces requirements[32]) описывают особенности взаимодействия разрабатываемой системы с другими системами и операционной средой.

Несколько простых, изолированных от контекста и друг от друга примеров требований к интерфейсам:

- *Обмен данными между клиентской и серверной частями приложения при осуществлении фоновых AJAX-запросов должен быть реализован в формате JSON.*
- *Протоколирование событий должно вестись в журнале событий операционной системы.*
- *Соединение с почтовым сервером должно выполняться согласно RFC3207 («SMTP over TLS»).*

Требования к данным (data requirements[33]) описывают структуры данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой системы. Часто сюда относят описание базы данных и особенностей её использования.

Несколько простых, изолированных от контекста и друг от друга примеров требований к данным:

- *Все данные системы, за исключением пользовательских документов, должны храниться в БД под управлением СУБД MySQL, пользовательские документы должны храниться в БД под управлением СУБД MongoDB.*
- *Информация о кассовых транзакциях за текущий месяц должна храниться в операционной таблице, а по завершении месяца переноситься в архивную.*
- *Для ускорения операций поиска по тексту статей и обзо-*

[32] **External interface requirements.** Requirements in this category describe the connections between the system and the rest of the universe. They include interfaces to users, hardware, and other software systems. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[33] **Data requirements.** Data requirement describe the format, data type, allowed values, or default value for a data element; the composition of a complex business data structure; or a report to be generated. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

ров должны быть предусмотрены полнотекстовые индексы на соответствующих полях таблиц.

Спецификация требований (software requirements specification, SRS[34]) объединяет в себе описание всех требований уровня продукта и может представлять собой весьма объёмный документ (сотни и тысячи страниц).

Поскольку требований может быть очень много, а их приходится не только единожды написать и согласовать между собой, но и постоянно обновлять, работу проектной команды по управлению требованиями значительно облегчают соответствующие инструментальные средства (requirements management tools[35, 36]).

Для более глубокого понимания принципов создания, организации и использования набора требований рекомендуется ознакомиться с фундаментальной работой Карла Вигерса «Разработка требований к программному обеспечению» (Wiegers K., Beatty J. Software Requirements. – 3rd ed. (Developer Best Practices). В этой же книге (в приложениях) приведены весьма наглядные учебные примеры документов, описывающих требования различного уровня.

1.5 Свойства качественных требований

В процессе тестирования требований проверяется их соответствие определённому набору свойств (рисунок 1.f).

Завершённость (completeness[37]). Требование является полным и законченным с точки зрения представления в нём всей необходимой информации, ничто не пропущено по соображениям «это и так всем понятно».

---

[34] **Software requirements specification, SRS.** SRS describes as fully as necessary the expected behavior of the software system. The SRS is used in development, testing, quality assurance, project management, and related project functions. People call this deliverable by many different names, including business requirements document, functional spec, requirements document, and others. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[35] **Requirements management tool.** A tool that supports the recording of requirements, requirements attributes (e.g. priority, knowledge responsible) and annotation, and facilitates traceability through layers of requirements and requirements change management. Some requirements management tools also provide facilities for static analysis, such as consistency checking and violations to predefined requirements rules. ISTQB Glossary.

[36] Обширный список инструментальных средств управления требованиями можно найти здесь. URL: http://makingofsoftware.com/resources/list-of-rm-tools.

[37] Each requirement must contain all the information necessary for the reader to understand it. In the case of functional requirements, this means providing the information the developer needs to be able to implement it correctly. No requirement or necessary information should be absent. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

Типичные проблемы с завершённостью:

- Отсутствуют нефункциональные составляющие требования или ссылки на соответствующие нефункциональные требования (например: «*пароли должны храниться в зашифрованном виде*» – каков алгоритм шифрования?).
- Указана лишь часть некоторого перечисления (например: «*экспорт осуществляется в форматы PDF, PNG и т. д.*» – что мы должны понимать под «и т. д.»?).
- Приведённые ссылки неоднозначны (например: «*см. выше*» вместо «*см. раздел 123.45.b*»).

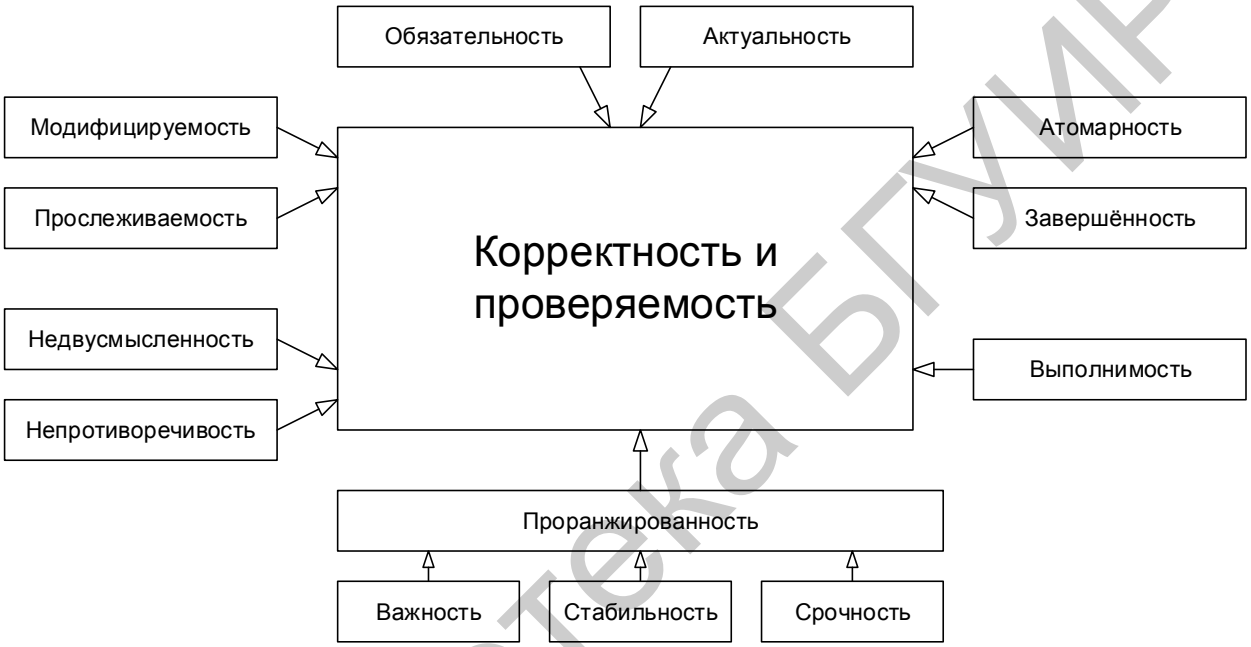


Рисунок 1.f – Свойства качественного требования

Атомарность, единичность (atomicity[38]). Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершённости и оно описывает одну и только одну ситуацию.

Типичные проблемы с атомарностью:

- В одном требовании фактически содержится несколько независимых (например: «*кнопка “Restart” не должна отображаться при остановленном сервисе, окно “Log” должно вмещать не менее 20-ти записей о последних действиях пользователя*» – здесь зачем-то в одном предложении описаны совершенно разные элементы интерфейса в совершенно разных контекстах).
- Требование допускает разночтение в силу грамматических особенностей языка (например: «*если пользователь подтвержда-*

[38] Each requirement you write represents a single market need that you either satisfy or fail to satisfy. A well written requirement is independently deliverable and represents an incremental increase in the value of your software. Blain T. Writing Good Requirements – The Big Ten Rules. URL: http://tynerblain.com/blog/2006/05/25/writing-good-requirements-the-big-ten-rules/.

ет заказ и редактирует заказ или откладывает заказ, должен выдаваться запрос на оплату» – здесь описаны три разных случая, и это требование стоит разбить на три отдельных во избежание путаницы). Такое нарушение атомарности часто влечёт за собой возникновение противоречивости.

- В одном требовании объединено описание нескольких независимых ситуаций (например: «*когда пользователь входит в систему, ему должно отображаться приветствие; когда пользователь вошёл в систему, должно отображаться имя пользователя; когда пользователь выходит из системы, должно отображаться прощание*» – все эти три ситуации заслуживают того, чтобы быть описанными отдельными и куда более детальными требованиями).

Непротиворечивость, последовательность (consistency[39]). Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.

Типичные проблемы с непротиворечивостью:

- Противоречия внутри одного требования (например: «*после успешного входа в систему пользователя, не имеющего права входить в систему…*» – тогда как он успешно вошёл в систему, если не имел такого права?).
- Противоречия между двумя и более требованиями, между таблицей и текстом, рисунком и текстом, требованием и прототипом и т. д. (например: «*712.a Кнопка “Close” всегда должна быть красной*» и «*36452.x Кнопка “Close” всегда должна быть синей*» – так всё же красной или синей?).
- Использование неверной терминологии или использование разных терминов для обозначения одного и того же объекта или явления (например: «*в случае, если разрешение окна составляет менее 800x600…*» – разрешение есть у экрана, у окна есть размер).

Недвусмысленность (unambiguousness[40], clearness). Требование описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок и допускает только однозначное объективное понимание. Требование атомарно в плане невозможности различной трактовки сочетания отдельных фраз.

[39] Consistent requirements don’t conflict with other requirements of the same type or with higher-level business, user, or system requirements. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[40] Natural language is prone to two types of ambiguity. One type I can spot myself, when I can think of more than one way to interpret a given requirement. The other type of ambiguity is harder to catch. That’s when different people read the requirement and come up with different interpretations of it. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

Типичные проблемы с недвусмысленностью:

- Использование терминов или фраз, допускающих субъективное толкование (например: «*приложение должно поддерживать передачу больших объёмов данных*» – насколько «больших»?). Вот лишь небольшой перечень слов и выражений, которые можно считать верными признаками двусмысленности: адекватно (adequate), быть способным (be able to), легко (easy), обеспечивать (provide for), как минимум (as a minimum), быть способным (be capable of), эффективно (effectively), своевременно (timely), применимо (as applicable), если возможно (if possible), будет определено позже (to be determined, TBD), по мере необходимости (as appropriate), если это целесообразно (if practical), но не ограничиваясь (but not limited to), быть способно (capability of), иметь возможность (capability to), нормально (normal), минимизировать (minimize), максимизировать (maximize), оптимизировать (optimize), быстро (rapid), удобно (user-friendly), просто (simple), часто (often), обычно (usual), большой (large), гибкий (flexible), устойчивый (robust), по последнему слову техники (state-of-the-art), улучшенный (improved), результативно (efficient). Вот утрированный пример требования, звучащего очень красиво, но совершенно нереализуемого и непонятного: «*В случае необходимости оптимизации передачи больших файлов система должна эффективно использовать минимум оперативной памяти, если это возможно*».
- Использование неочевидных или двусмысленных аббревиатур без расшифровки (например: «*доступ к ФС осуществляется посредством системы прозрачного шифрования*» и «*ФС предоставляет возможность фиксировать сообщения в их текущем состоянии с хранением истории всех изменений*» – ФС здесь обозначает файловую систему? Точно? А не какой-нибудь «Фиксатор Сообщений»?).
- Формулировка требований из соображений, что нечто должно быть всем очевидно (например: «*Система конвертирует входной файл из формата PDF в выходной файл формата PNG*» – и при этом автор считает совершенно очевидным, что имена файлов система получает из командной строки, а многостраничный PDF конвертируется в несколько PNG-файлов, к именам которых добавляется «page-1», «page-2» и т. д.). Эта проблема перекликается с нарушением корректности.

Выполнимость (feasibility[41]). Требование технологически выполнимо и может быть реализовано в рамках бюджета и сроков разработки проекта.

[41] It must be possible to implement each requirement within the known capabilities and limitations of the system and its operating environment, as well as within project constraints of time, budget, and staff. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

Типичные проблемы с выполнимостью:

- Так называемое «озолочение» (gold plating) – требования, которые крайне долго и/или дорого реализуются и при этом практически бесполезны для конечных пользователей (например: «*настройка параметров для подключения к базе данных должна поддерживать распознавание символов из жестов, полученных с устройств трёхмерного ввода*»).
- Технически нереализуемые на современном уровне развития технологий требования (например: «*анализ договоров должен выполняться с применением искусственного интеллекта, который будет выносить однозначное корректное заключение о степени выгоды от заключения договора*»).
- В принципе нереализуемые требования (например: «*система поиска должна заранее предусматривать все возможные варианты поисковых запросов и кэшировать их результаты*»).

Обязательность, нужность (obligation[42]) и **актуальность** (up-to-date). Если требование не является обязательным к реализации, оно должно быть просто исключено из набора требований. Если требование нужное, но «не очень важное», для указания этого факта используется указание приоритета (см. «проранжированность по…»). Также исключены (или переработаны) должны быть требования, утратившие актуальность.

Типичные проблемы с обязательностью и актуальностью:

- Требование было добавлено «на всякий случай», хотя реальной потребности в нём не было и нет.
- Требованию выставлены неверные значения приоритета по критериям важности и/или срочности.
- Требование устарело, но не было переработано или удалено.

Прослеживаемость (traceability[43, 44]). Прослеживаемость бывает вертикальной (vertical traceability[45]) и горизонтальной (horizontal traceabil-

[42] Each requirement should describe a capability that provides stakeholders with the anticipated business value, differentiates the product in the marketplace, or is required for conformance to an external standard, policy, or regulation. Every requirement should originate from a source that has the authority to provide requirements. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[43] **Traceability.** The ability to identify related items in documentation and software, such as requirements with associated tests. ISTQB Glossary.

[44] A traceable requirement can be linked both backward to its origin and forward to derived requirements, design elements, code that implements it, and tests that verify its implementation. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[45] **Vertical traceability.** The tracing of requirements through the layers of development documentation to components. ISTQB Glossary.

ity[46]). Вертикальная позволяет соотносить между собой требования на различных уровнях требований, горизонтальная позволяет соотносить требование с тест-планом, тест-кейсами, архитектурными решениями и т. д.

Для обеспечения прослеживаемости часто используются специальные инструменты по управлению требованиями (requirements management tool[47]) и/или матрицы прослеживаемости (traceability matrix[48]).

Типичные проблемы с прослеживаемостью:

- Требования не пронумерованы, не структурированы, не имеют оглавления, не имеют работающих перекрёстных ссылок.
- При разработке требований не были использованы инструменты и техники управления требованиями.
- Набор требований неполный, носит обрывочный характер с явными «пробелами».

Модифицируемость (modifiability[49]). Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований. Можно говорить о наличии модифицируемости в том случае, если при доработке требований искомую информацию легко найти, а её изменение не приводит к нарушению иных описанных в этом перечне свойств.

Типичные проблемы с модифицируемостью:

- Требования неатомарны (см. «атомарность») и непрослеживаемы (см. «прослеживаемость»), а потому их изменение с высокой вероятностью порождает противоречивость (см. «непротиворечивость»).
- Требования изначально противоречивы (см. «непротиворечивость»). В такой ситуации внесение изменений (не связанных с устранением противоречивости) только усугубляет ситуацию, увеличивая противоречивость и снижая прослеживаемость.

[46] **Horizontal traceability.** The tracing of requirements for a test level through the layers of test documentation (e.g. test plan, test design specification, test case specification and test procedure specification or test script). ISTQB Glossary.

[47] **Requirements management tool.** A tool that supports the recording of requirements, requirements attributes (e.g. priority, knowledge responsible) and annotation, and facilitates traceability through layers of requirements and requirements change management. Some requirements management tools also provide facilities for static analysis, such as consistency checking and violations to predefined requirements rules. ISTQB Glossary.

[48] **Traceability matrix.** A two-dimensional table, which correlates two entities (e.g., requirements and test cases). The table allows tracing back and forth the links of one entity to the other, thus enabling the determination of coverage achieved and the assessment of impact of proposed changes. ISTQB Glossary.

[49] To facilitate modifiability, avoid stating requirements redundantly. Repeating a requirement in multiple places where it logically belongs makes the document easier to read but harder to maintain. The multiple instances of the requirement all have to be modified at the same time to avoid generating inconsistencies. Cross-reference related items in the SRS to help keep them synchronized when making changes. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

- Требования представлены в неудобной для обработки форме (например, не использованы инструменты управления требованиями, и в итоге команде приходится работать с десятками огромных текстовых документов).

Проранжированность по важности, стабильности, срочности (ranked[50] for importance, stability, priority). Важность характеризует зависимость успеха проекта от успеха реализации требования. Стабильность характеризует вероятность того, что в обозримом будущем в требование не будет внесено никаких изменений. Срочность определяет распределение во времени усилий проектной команды по реализации того или иного требования.

Типичные проблемы с проранжированностью состоят в её отсутствии или неверной реализации и приводят к следующим последствиям.

- Проблемы с проранжированностью по важности повышают риск неверного распределения усилий проектной команды, направления усилий на второстепенные задачи и конечного провала проекта из-за неспособности продукта выполнять ключевые задачи с соблюдением ключевых условий.
- Проблемы с проранжированностью по стабильности повышают риск выполнения бессмысленной работы по совершенствованию, реализации и тестированию требований, которые в самое ближайшее время могут претерпеть кардинальные изменения (вплоть до полной утраты актуальности).
- Проблемы с проранжированностью по срочности повышают риск нарушения желаемой заказчиком последовательности реализации функциональности и ввода этой функциональности в эксплуатацию.

Корректность (correctness[51]) и **проверяемость** (verifiability[52]). Фактически эти свойства вытекают из соблюдения всех вышеперечисленных (или можно сказать, что они не выполняются, если нарушено хотя бы одно из вышеперечисленных). В дополнение можно отметить, что проверяемость подразумевает возможность создания объективного тест-кейса (тест-кейсов), однозначно показывающего, что требование реализовано верно и поведение приложения в точности соответствует требованию.

---

[50] Prioritize business requirements according to which are most important to achieving the desired value. Assign an implementation priority to each functional requirement, user requirement, use case flow, or feature to indicate how essential it is to a particular product release. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[51] Each requirement must accurately describe a capability that will meet some stakeholder's need and must clearly describe the functionality to be built. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

[52] If a requirement isn't verifiable, deciding whether it was correctly implemented becomes a matter of opinion, not objective analysis. Requirements that are incomplete, inconsistent, infeasible, or ambiguous are also unverifiable. Wiegers K., Beatty J. Software Requirements. – 3rd ed.

К типичным проблемам с корректностью также можно отнести:

- Опечатки (особенно опасны опечатки в аббревиатурах, превращающие одну осмысленную аббревиатуру в другую также осмысленную, но не имеющую отношения к некоему контексту; такие опечатки крайне сложно заметить).
- Наличие неаргументированных требований к дизайну и архитектуре.
- Плохое оформление текста и сопутствующей графической информации, грамматические, пунктуационные и иные ошибки в тексте.
- Неверный уровень детализации (например, слишком глубокая детализация требования на уровне бизнес-требований или недостаточная детализация на уровне требований к продукту).
- Требования к пользователю, а не к приложению (например: «пользователь должен быть в состоянии отправить сообщение» – увы, мы не можем влиять на состояние пользователя).

1.6 Техники тестирования требований

Тестирование документации и требований относится к разряду нефункционального тестирования (non-functional testing[53]). Основные техники такого тестирования в контексте требований таковы.

Взаимный просмотр (peer review[54]). Взаимный просмотр («рецензирование») является одной из наиболее активно используемых техник тестирования требований и может быть представлен в одной из трёх следующих форм (по мере нарастания его сложности и цены):

- **Беглый просмотр** (walkthrough[55]) может выражаться как в показе автором своей работы коллегам с целью создания общего понимания и получения обратной связи, так и в простом обмене результатами работы между двумя и более авторами с тем, чтобы коллега высказал свои вопросы и замечания. Это самый быстрый и наиболее широко используемый вид просмотра.

  Для запоминания: аналог беглого просмотра – это ситуация, когда вы в школе с одноклассниками проверяли перед сдачей сочинения друг друга, чтобы найти описки и ошибки.

[53] **Non-functional testing.** Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, maintainability and portability. ISTQB Glossary.

[54] **Peer review.** A review of a software work product by colleagues of the producer of the product for the purpose of identifying defects and improvements. Examples are inspection, technical review and walkthrough. ISTQB Glossary.

[55] **Walkthrough.** A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content. ISTQB Glossary.

- **Технический просмотр** (technical review[56]) выполняется группой специалистов. В идеальной ситуации каждый специалист должен представлять свою область знаний. Просматриваемый продукт не может считаться достаточно качественным, пока хотя бы у одного просматривающего остаются замечания.

  Для запоминания: аналог технического просмотра – это ситуация, когда некий договор визирует юридический отдел, бухгалтерия и т. д.

- **Формальная инспекция** (inspection[57]) представляет собой структурированный, систематизированный и документируемый подход к анализу документации. Для его выполнения привлекается большое количество специалистов, само выполнение занимает достаточно много времени, и потому этот вариант просмотра используется достаточно редко (как правило, при получении на сопровождение и доработку проекта, созданием которого ранее занималась другая компания).

  Для запоминания: аналог формальной инспекции – это ситуация генеральной уборки квартиры (включая содержимое всех шкафов, холодильника, кладовки и т. д.).

Вопросы. Следующей очевидной техникой тестирования и повышения качества требований является (повторное) использование техник выявления требований, а также (как отдельный вид деятельности) – задавание вопросов. Если хоть что-то в требованиях вызывает у вас непонимание или подозрение – задавайте вопросы. Можно спросить представителей заказчика, можно обратиться к справочной информации. По многим вопросам можно обратиться к более опытным коллегам при условии, что у них имеется соответствующая информация, ранее полученная от заказчика. Главное, чтобы ваш вопрос был сформулирован таким образом, чтобы полученный ответ позволил улучшить требования.

Поскольку здесь начинающие тестировщики допускают много ошибок, рассмотрим подробнее. В таблице 1.a приведено несколько плохо сформулированных требований, а также примеров плохих и хороших вопросов. Плохие вопросы провоцируют на бездумные ответы, не содержащие полезной информации.

[56] **Technical review.** A peer group discussion activity that focuses on achieving consensus on the technical approach to be taken. ISTQB Glossary.

[57] **Inspection.** A type of peer review that relies on visual examination of documents to detect defects, e.g. violations of development standards and non-conformance to higher level documentation. The most formal review technique and therefore always based on a documented procedure. ISTQB Glossary.

Таблица 1.а – Пример плохих и хороших вопросов к требованиям

Плохое требование	**Плохие вопросы**	**Хорошие вопросы**
«Приложение должно быстро запускаться»	«Насколько быстро?» (На это вы рискуете получить ответы в стиле «очень быстро», «максимально быстро», «нууу… просто быстро».) «А если не получится быстро?» (Этим вы рискуете просто удивить или даже разозлить заказчика.) «Всегда?» («Да, всегда». Хм, а вы ожидали другого ответа?)	«Каково максимально допустимое время запуска приложения, на каком оборудовании и при какой загруженности этого оборудования операционной системой и другими приложениями? На достижение каких целей влияет скорость запуска приложения? Допускается ли фоновая загрузка отдельных компонентов приложения? Что является критерием того, что приложение закончило запуск?»
«Опционально должен поддерживаться экспорт документов в формат PDF»	«Любых документов?» (Ответы «да, любых» или «нет, только открытых» вам всё равно не помогут.) «В PDF какой версии должен производиться экспорт?» (Сам по себе вопрос хорош, но он не даёт понять, что имелось в виду под «опционально».) «Зачем?» («Нужно!» Именно так хочется ответить, если вопрос не раскрыт полностью.)	«Насколько возможность экспорта в PDF важна? Как часто, кем и с какой целью она будет использоваться? Является ли PDF единственным допустимым форматом для этих целей или есть альтернативы? Допускается ли использование внешних утилит (например, виртуальных PDF-принтеров) для экспорта документов в PDF?»

Продолжение таблицы 1.а

Плохое требование	**Плохие вопросы**	**Хорошие вопросы**
«Если дата события не указана, она выбирается автоматически»	«А если указана?» (То она указана. Логично, не так ли?) «А если дату невозможно выбрать автоматически?» (Сам вопрос интересен, но без пояснения причин невозможности звучит как насмешка.) «А если у события нет даты?» (Тут автор вопроса, скорее всего, хотел уточнить, обязательно ли это поле для заполнения. Но из самого требования видно, что обязательно: если оно не заполнено человеком, его должен заполнить компьютер.)	«Возможно, имелось в виду, что дата **генерируется** автоматически, а не **выбирается**? Если да, то по какому алгоритму она генерируется? Если нет, то из какого набора выбирается дата и как генерируется этот набор? P.S. Возможно, стоит использовать текущую дату?»

Тест-кейсы и чек-листы. Мы помним, что хорошее требование является проверяемым, а значит, должны существовать объективные способы определения того, верно ли реализовано требование. Продумывание чек-листов или даже полноценных тест-кейсов в процессе анализа требований позволяет нам определить, насколько требование проверяемо. Если вы можете быстро придумать несколько пунктов чек-листа, это ещё не признак того, что с требованием всё хорошо (например, оно может противоречить каким-то другим требованиям). Но если никаких идей по тестированию требования в голову не приходит – это тревожный знак.

Рекомендуется для начала убедиться, что вы понимаете требование (в том числе прочесть соседние требования, задать вопросы коллегам и т. д.) Также можно пока отложить работу с данным конкретным требованием и вернуться к нему позднее – возможно, анализ других требований позволит вам лучше понять и это «конкретное». Но если ничто не помогает – скорее всего, с требованием что-то не так.

Справедливости ради надо отметить, что на начальном этапе проработки требований такие случаи встречаются очень часто – требова-

ния сформированы очень поверхностно, расплывчато и явно нуждаются в доработке, т. е. здесь нет необходимости проводить сложный анализ, чтобы констатировать непроверяемость требования.

На стадии же, когда требования уже хорошо сформулированы и протестированы, вы можете продолжать использовать эту технику, совмещая разработку тест-кейсов и дополнительное тестирование требований.

Исследование поведения системы. Эта техника логически вытекает из предыдущей (продумывания тест-кейсов и чек-листов), но отличается тем, что здесь тестированию подвергается, как правило, не одно требование, а целый набор. Тестировщик мысленно моделирует процесс работы пользователя с системой, созданной по тестируемым требованиям, и ищет неоднозначные или вовсе неописанные варианты поведения системы. Этот подход сложен, требует достаточной квалификации тестировщика, но способен выявить нетривиальные недоработки, которые почти невозможно заметить, тестируя требования по отдельности.

Рисунки (графическое представление). Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы, интеллект-карты[58] и т. д. Графическое представление удобно одновременно своей наглядностью и краткостью (например, UML-схема базы данных, занимающая один экран, может быть описана несколькими десятками страниц текста). На рисунке очень легко заметить, что какие-то элементы «не стыкуются», что где-то чего-то не хватает и т. д. Если вы для графического представления требований будете использовать общепринятую нотацию (например, уже упомянутый UML), вы получите дополнительные преимущества: вашу схему смогут без труда понимать и дорабатывать коллеги, а в итоге может получиться хорошее дополнение к текстовой форме представления требований.

Прототипирование. Можно сказать, что прототипирование часто является следствием создания графического представления и анализа поведения системы. С использованием специальных инструментов можно очень быстро сделать наброски пользовательских интерфейсов, оценить применимость тех или иных решений и даже создать не просто «прототип ради прототипа», а заготовку для дальнейшей разработки, если окажется, что реализованное в прототипе (возможно, с небольшими доработками) устраивает заказчика.

[58] Mind map. URL: http://en.wikipedia.org/wiki/Mind_map.

1.7 Контрольные вопросы и задания

- Сформулируйте определение требования.
- Кто является основным источником и потребителем требований?
- Какова связь требований и архитектуры проекта?
- Опишите уровни требований – в чём они выражаются и что описывают?
- Какие типы требований вы знаете?
- Перечислите свойства хорошего требования.
- Приведите примеры плохих и хороших вопросов к требованиям.
- Какими свойствами должны обладать наборы требований?
- Какие проблемы чаще всего возникают при работе с требованиями? В чём их суть?
- Назовите основные пути выявления требований.
- Перечислите характеристики хороших наборов требований.
- Назовите основные проблемы с наборами требований.
- Каковы преимущества и недостатки анкетирования как способа определения требований?
- Каковы преимущества и недостатки наблюдения как способа определения требований?
- Каковы преимущества и недостатки самостоятельного определения требований на основе документов?
- Каковы преимущества и недостатки семинаров как способа определения требований?
- Каковы преимущества и недостатки прототипирования как способа определения требований?
- Перечислите основные источники требований.
- Сделайте сравнительный анализ основных техник выявления требований.
- Какие требования относятся к группе нефункциональных требований?
- Какие требования относятся к группе функциональных требований?
- Назовите основную цель функционального тестирования.
- Назовите основные техники тестирования требований.
- Чем отличается проектная документация от продуктной?
- Перечислите основные виды тестирования. Сформулируйте их определения.
- Перечислите основные уровни тестирования. Дайте им определения.
- Какие виды документации можно тестировать? Перечислите.

2 ЧЕК-ЛИСТЫ, ТЕСТ-КЕЙСЫ, НАБОРЫ ТЕСТ-КЕЙСОВ

2.1 Чек-листы

Тестировщику приходится работать с огромным количеством информации, выбирать из множества вариантов решения задач и изобретать новые. В процессе этой деятельности объективно невозможно удержать в голове все мысли, а потому продумывание и разработку тест-кейсов рекомендуется выполнять с использованием так называемых «чек-листов».

Чек-лист (checklist[59]) – набор идей [тест-кейсов]. Последнее слово не зря взято в скобки, т. к. в общем случае чек-лист – это просто набор идей: идей по тестированию, идей по разработке, идей по планированию и управлению – **любых** идей.

Чек-лист чаще всего представляет собой обычный и привычный нам список, который может быть:

- Списком, в котором последовательность пунктов не имеет значения (например, список значений некоего поля).
- Списком, в котором последовательность пунктов важна (например, шаги в краткой инструкции).
- Структурированным (многоуровневым) списком (вне зависимости от учёта последовательности пунктов), что позволяет отразить иерархию идей.

Важно понять, что нет и не может быть никаких запретов и ограничений при разработке чек-листов – главное, чтобы они помогали в работе. Иногда чек-листы могут даже выражаться графически (например, с использованием ментальных карт[60] или концепт-карт[61]), хотя традиционно их составляют в виде многоуровневых списков.

Поскольку в разных проектах встречаются однотипные задачи, хорошо продуманные и аккуратно оформленные чек-листы могут использоваться повторно, чем достигается экономия сил и времени.

Для того чтобы чек-лист был действительно полезным инструментом, он должен обладать рядом важных свойств.

Логичность. Чек-лист пишется не «просто так», а на основе целей

[59] Понятие «чек-листа» не завязано на тестирование как таковое – это совершенно универсальная техника, которая может применяться в любой без исключения области жизни. В русском языке вне контекста информационных технологий чаще используется понятное и привычное слово «список» (например, «список покупок», «список дел» и т. д.), но в тестировании прижилась калькированная с английского версия – «чек-лист».

[60] Mind map. URL: http://en.wikipedia.org/wiki/Mind_map.

[61] Concept map. URL: http://en.wikipedia.org/wiki/Concept_map.

и для того, чтобы помочь в достижении этих целей. К сожалению, одной из самых частых и опасных ошибок при составлении чек-листа является превращение его в свалку мыслей, которые никак не связаны друг с другом.

Последовательность и структурированность. Со структурированностью всё достаточно просто – она достигается за счёт оформления чек-листа в виде многоуровневого списка. Что до последовательности, то даже в том случае, когда пункты чек-листа не описывают цепочку действий, человеку всё равно удобнее воспринимать информацию в виде неких небольших групп идей, переход между которыми является понятным и очевидным (например, сначала можно прописать идеи простых позитивных тест-кейсов SPECIAL_TERMINS_Positive_Testing, потом идеи простых негативных тест-кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит писать эти идеи вперемешку).

Полнота и неизбыточность. Чек-лист должен представлять собой аккуратную «сухую выжимку» идей, в которых нет дублирования (часто появляется из-за разных формулировок одной и той же идеи), и в то же время ничто важное не упущено.

Правильно создавать и оформлять чек-листы также помогает восприятие их не только как хранилища наборов идей, но как «требования для составления тест-кейсов». Эта мысль приводит к пересмотру и переосмыслению свойств качественных требований (см. подраздел 1.5) в применении к чек-листам.

Задание 2.а: перечитайте подраздел 1.5 и подумайте, какие свойства качественных требований можно также считать и свойствами качественных чек-листов?

Итак, рассмотрим процесс создания чек-листа.

Поскольку мы не можем сразу «протестировать всё приложение» (это слишком большая задача, чтобы решить её одним махом), нам уже сейчас нужно выбрать некую логику построения чек-листов – да, их будет несколько (в итоге их можно будет структурированно объединить в один, но это не обязательно).

Типичными вариантами такой логики является создание отдельных чек-листов для:

- различных уровней функционального тестирования;
- отдельных частей (модулей и подмодулей) приложения (см. «Модуль и подмодуль приложения» в подразделе 2.3);
- отдельных требований, групп требований, уровней и типов требований (см. подраздел 1.4);
- типичных пользовательских сценариев (см. «Пользовательские сценарии (сценарии использования)» в подразделе 2.5);

- частей или функций приложения, наиболее подверженных рискам.

Этот список можно расширять и дополнять, можно комбинировать его пункты, получая, например, чек-листы для проверки наиболее типичных сценариев, затрагивающих некую часть приложения.

Чтобы проиллюстрировать принципы построения чек-листов, мы воспользуемся логикой разбиения функций приложения по степени их важности (классификацию по убыванию степени важности функций приложения) на три категории:

- Базовые функции, без которых существование приложения теряет смысл (т. е. самые важные – то, ради чего приложение вообще создавалось), или нарушение работы которых создаёт объективные серьёзные проблемы для среды исполнения.
- Функции, востребованные большинством пользователей в их повседневной работе.
- Остальные функции (разнообразные «мелочи», проблемы с которыми не сильно повлияют на ценность приложения для конечного пользователя).

Рассмотрим функции, без которых существование приложения теряет смысл. Сначала приведём целиком весь чек-лист для дымового тестирования, а потом рассмотрим его подробнее:

- Конфигурирование и запуск.
- Обработка файлов (таблица 2.а).

Таблица 2.а – Обработка файлов

Кодировки входных файлов	Форматы входных файлов		
	TXT	HTML	MD
WIN1251	+	+	+
CP866	+	+	+
KOI8R	+	+	+

- Остановка.

Здесь перечислены все ключевые функции приложения.

Конфигурирование и запуск. Если приложение невозможно настроить для работы в пользовательской среде, оно бесполезно. Если приложение не запускается, оно бесполезно. Если на стадии запуска возникают проблемы, они могут негативно отразиться на функционировании приложения и потому также заслуживают пристального внимания.

Примечание – В нашем примере мы столкнулись со скорее нетипичным случаем – приложение конфигурируется параметрами командной строки, а потому разделить операции «конфигурирования» и «запуска» не представляется возможным; в реальной жизни для подавляюще-

го большинства приложений эти операции выполняются раздельно.

Обработка файлов. Приложение разрабатывалось для обработки файлов, потому здесь, даже на стадии создания чек-листа, мы не поленились создать матрицу, отражающую все возможные комбинации допустимых форматов и допустимых кодировок входных файлов, чтобы ничего не забыть и подчеркнуть важность соответствующих проверок.

Остановка. С точки зрения пользователя эта функция может не казаться столь уж важной, но остановка (и запуск) любого приложения связана с большим количеством системных операций, проблемы с которыми могут привести к множеству серьёзных последствий (вплоть до невозможности повторного запуска приложения или нарушения работы операционной системы).

Рассмотрим функции, востребованные большинством пользователей. Следующим шагом мы будем выполнять проверку того, как приложение ведёт себя в обычной повседневной жизни, пока не затрагивая «экзотические» ситуации. Очень частым вопросом является допустимость дублирования проверок на разных уровнях функционального тестирования – можно ли так делать? Одновременно и «нет», и «да». «Нет» – в том смысле, что не допускается (не имеет смысла) повторение тех же проверок, которые только что были выполнены. «Да» – в том смысле, что любую проверку можно детализировать и снабдить дополнительными деталями:

- Конфигурирование и запуск:
  - С верными параметрами:
    - Значения SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME указаны и содержат пробелы и кириллические символы (повторить для форматов путей в Windows- и *nix-файловых системах, обратить внимание на имена логических дисков и разделители имён каталогов (“/” и “\”)).
    - Значение LOG_FILE_NAME не указано.
  - Без параметров.
  - С недостаточным количеством параметров.
  - С неверными параметрами:
    - Недопустимый путь SOURCE_DIR.
    - Недопустимый путь DESTINATION_DIR.
    - Недопустимое имя LOG_FILE_NAME.
    - DESTINATION_DIR находится внутри SOURCE_DIR.
    - Значения DESTINATION_DIR и SOURCE_DIR совпадают.
- Обработка файлов:
  - Разные форматы, кодировки и размеры (таблица 2.b).

Таблица 2.b – Разные форматы и кодировки входных файлов

Кодировки входных файлов	Форматы входных файлов		
	TXT	HTML	MD
WIN1251	100 КБ	50 МБ	10 МБ
CP866	10 МБ	100 КБ	50 МБ
KOI8R	50 МБ	10 МБ	100 КБ
Любая	0 Б		
Любая	50 МБ + 1 Б	50 МБ + 1 Б	50 МБ + 1 Б
–	Любой недопустимый формат		
Любая	Повреждения в допустимом формате		

    - Недоступные входные файлы:
        - Нет прав доступа.
        - Файл открыт и заблокирован.
        - Файл с атрибутом «только для чтения».
- Остановка:
    - Закрытием окна консоли.
- Журнал работы приложения:
    - Автоматическое создание (при отсутствии журнала).
    - Продолжение (дополнение журнала) при повторных запусках.
- Производительность:
    - Элементарный тест с грубой оценкой.

Обратите внимание, что чек-лист может содержать не только «предельно сжатые тезисы», но и вполне развёрнутые комментарии, если это необходимо – лучше пояснить суть идеи подробнее, чем потом гадать, что же имелось в виду.

Также обратите внимание, что многие пункты чек-листа носят весьма высокоуровневый характер, и это нормально. Например, «повреждения в допустимом формате» (см. матрицу с кодировками, форматами и размерами) звучит расплывчато, но этот недочёт будет устранён уже на уровне полноценных тест-кейсов.

Рассмотрим остальные функции и особые сценарии. Пришло время обратить внимание на разнообразные мелочи и «хитрые» нюансы, проблемы с которыми едва ли сильно озаботят пользователя, но формально всё же будут считать ошибками:

- Конфигурирование и запуск:
    - Значения SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME:
        - В разных стилях (Windows-пути + *nix-пути) – одно в одном стиле, другое – в другом.
        - С использованием UNC-имён.

        - LOG_FILE_NAME внутри SOURCE_DIR.
        - LOG_FILE_NAME внутри DESTINATION_DIR.
    - Размер LOG_FILE_NAME на момент запуска:
        - 2-4 ГБ.
        - 4+ ГБ.
    - Запуск двух и более копий приложения с:
        - Одинаковыми параметрами SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME.
        - Одинаковыми SOURCE_DIR и LOG_FILE_NAME, но разными DESTINATION_DIR.
        - Одинаковыми DESTINATION_DIR и LOG_FILE_NAME, но разными SOURCE_DIR.
- Обработка файлов:
    - Файл верного формата, в котором текст представлен в двух и более поддерживаемых кодировках одновременно.
    - Размер входного файла:
        - 2-4 ГБ.
        - 4+ ГБ.

Задание 2.b: возможно, вам захотелось что-то изменить в приведённых выше чек-листах – это совершенно нормально и справедливо: нет и не может быть некоего «единственно верного идеального чек-листа» и ваши идеи вполне имеют право на существование, потому составьте свой собственный чек-лист или отметьте недостатки, которые вы заметили в приведённом выше чек-листе.

Как мы увидим в подразделе 2.2, создание качественного тест-кейса может потребовать длительной кропотливой и достаточно монотонной работы, которая при этом не требует от опытного тестировщика немалых интеллектуальных усилий, а потому переключение между работой над чек-листами (творческая составляющая) и расписыванием их в тест-кейсы (механическая составляющая) позволяет разнообразить рабочий процесс и снизить утомляемость. Хотя, конечно, написание сложных и притом качественных тест-кейсов может оказаться ничуть не менее творческой работой, чем продумывание чек-листов.

2.2 Тест-кейсы

Терминология и общие сведения

Для начала определимся с терминологией, поскольку здесь есть много путаницы, вызванной разными переводами англоязычных терминов на русский язык и разными традициями в тех или иных странах, фирмах и отдельных командах.

Во главе всего лежит термин «тест». Официальное определение звучит так.

!!!	**Тест** (test[62]) – набор из одного или нескольких тест-кейсов.

Поскольку среди всех прочих терминов этот легче и быстрее всего произносить, в зависимости от контекста под ним могут понимать и отдельный пункт чек-листа, и отдельный шаг в тест-кейсе, и сам тест-кейс, и набор тест-кейсов и т. д. Главное здесь одно: если вы слышите или видите слово «тест», воспринимайте его в контексте.

Теперь рассмотрим самый главный для нас термин – «тест-кейс».

!!!	**Тест-кейс** (test case[63]) – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства. Под тест-кейсом также может пониматься соответствующий документ, представляющий формальную запись тест-кейса.

Критически важно понять и запомнить: если у тест-кейса не указаны входные данные, условия выполнения и ожидаемые результаты и/или не ясна цель тест-кейса – это плохой тест-кейс (иногда он не имеет смысла, иногда его и вовсе невозможно выполнить).

Примечание – Иногда термин «test case» на русский язык переводят как «тестовый случай». Это вполне адекватный перевод, но из-за того, что «тест-кейс» удобнее произносить, наибольшее распространение получил именно этот вариант.

....	Остальные термины, связанные с тестами, тест-кейсами и тестовыми сценариями, на данном этапе можно прочитать просто в ознакомительных целях. Если вы откроете ISTQB-глоссарий на букву «Т», вы увидите огромное количество терминов, тесно связанных друг с другом перекрёстными ссылками: на начальном этапе изучения тестирования нет необходимости глубоко рассматривать их все, однако некоторые всё же заслуживают внимания. Они представлены ниже.

[62] **Test.** A set of one or more test cases. ISTQB Glossary.

[63] **Test case.** A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. ISTQB Glossary.

Высокоуровневый тест-кейс (high level test case[64]) – тест-кейс без конкретных входных данных и ожидаемых результатов.

Как правило, ограничивается общими идеями и операциями, схож по своей сути с подробно описанным пунктом чек-листа. Достаточно часто встречается в интеграционном тестировании и системном тестировании, а также на уровне дымового тестирования. Может служить отправной точкой для проведения исследовательского тестирования или для создания низкоуровневых тест-кейсов.

Низкоуровневый тест-кейс (low level test case[65]) – тест-кейс с конкретными входными данными и ожидаемыми результатами.

Представляет собой «полностью готовый к выполнению» тест-кейс и вообще является наиболее классическим видом тест-кейсов. Начинающих тестировщиков чаще всего учат писать именно такие тесты, т. к. прописать все данные подробно – намного проще, чем понять, какой информацией можно пренебречь, при этом не снизив ценность тест-кейса.

Спецификация тест-кейса (test case specification[66]) – документ, описывающий набор тест-кейсов (включая их цели, входные данные, условия и шаги выполнения, ожидаемые результаты) для тестируемого элемента (test item[67], test object[68]).

Спецификация теста (test specification[69]) – документ, состоящий из спецификации тест-дизайна (test design specification[70]), спецификации тест-кейса (test case specification[66]) и/или спецификации тест-процедуры (test procedure specification[71]).

Тест-сценарий (test scenario[72], test procedure specification, test script) – документ, описывающий последовательность действий по выполнению теста (также известен как «тест-скрипт»).

---

[64] **High level test case (logical test case).** A test case without concrete (implementation level) values for input data and expected results. Logical operators are used; instances of the actual values are not yet defined and/or available. ISTQB Glossary.

[65] **Low level test case.** A test case with concrete (implementation level) values for input data and expected results. Logical operators from high level test cases are replaced by actual values that correspond to the objectives of the logical operators. ISTQB Glossary.

[66] **Test case specification.** A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item. ISTQB Glossary.

[67] **Test item.** The individual element to be tested. There usually is one test object and many test items. ISTQB Glossary.

[68] **Test object.** The component or system to be tested. ISTQB Glossary.

[69] **Test specification.** A document that consists of a test design specification, test case specification and/or test procedure specification. ISTQB Glossary.

[70] **Test design specification.** A document specifying the test conditions (coverage items) for a test item, the detailed test approach and identifying the associated high level test cases. ISTQB Glossary.

[71] **Test procedure specification (test procedure).** A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script. ISTQB Glossary.

[72] **Test scenario.** A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script. ISTQB Glossary.

Внимание! Из-за особенностей перевода очень часто под термином «тест-сценарий» («тестовый сценарий») имеют в виду набор тест-кейсов.

Цель написания тест-кейсов

Тестирование можно проводить и без тест-кейсов (не нужно, но можно; да, эффективность такого подхода варьируется в очень широком диапазоне в зависимости от множества факторов). Наличие же тест-кейсов позволяет:

- Структурировать и систематизировать подход к тестированию (без чего крупный проект почти гарантированно обречён на провал).
- Вычислять метрики тестового покрытия (test coverage[73] metrics) и принимать меры по его увеличению (тест-кейсы здесь являются главным источником информации, без которого существование подобных метрик теряет смысл).
- Отслеживать соответствие текущей ситуации плану (сколько примерно понадобится тест-кейсов, сколько уже есть, сколько выполнено из запланированного на данном этапе количества и т. д.).
- Уточнить взаимопонимание между заказчиком, разработчиками и тестировщиками (тест-кейсы зачастую намного более наглядно показывают поведение приложения, чем это отражено в требованиях).
- Хранить информацию для длительного использования и обмена опытом между сотрудниками и командами (или как минимум – не пытаться удержать в голове сотни страниц текста).
- Проводить регрессионное тестирование и повторное тестирование (которые без тест-кейсов было бы вообще невозможно выполнить).
- Повышать качество требований (мы это уже рассматривали («Тест-кейсы и чек-листы» в подразделе 1.6): написание чек-листов и тест-кейсов – хорошая техника тестирования требований).
- Быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

2.3 Атрибуты (поля) тест-кейса

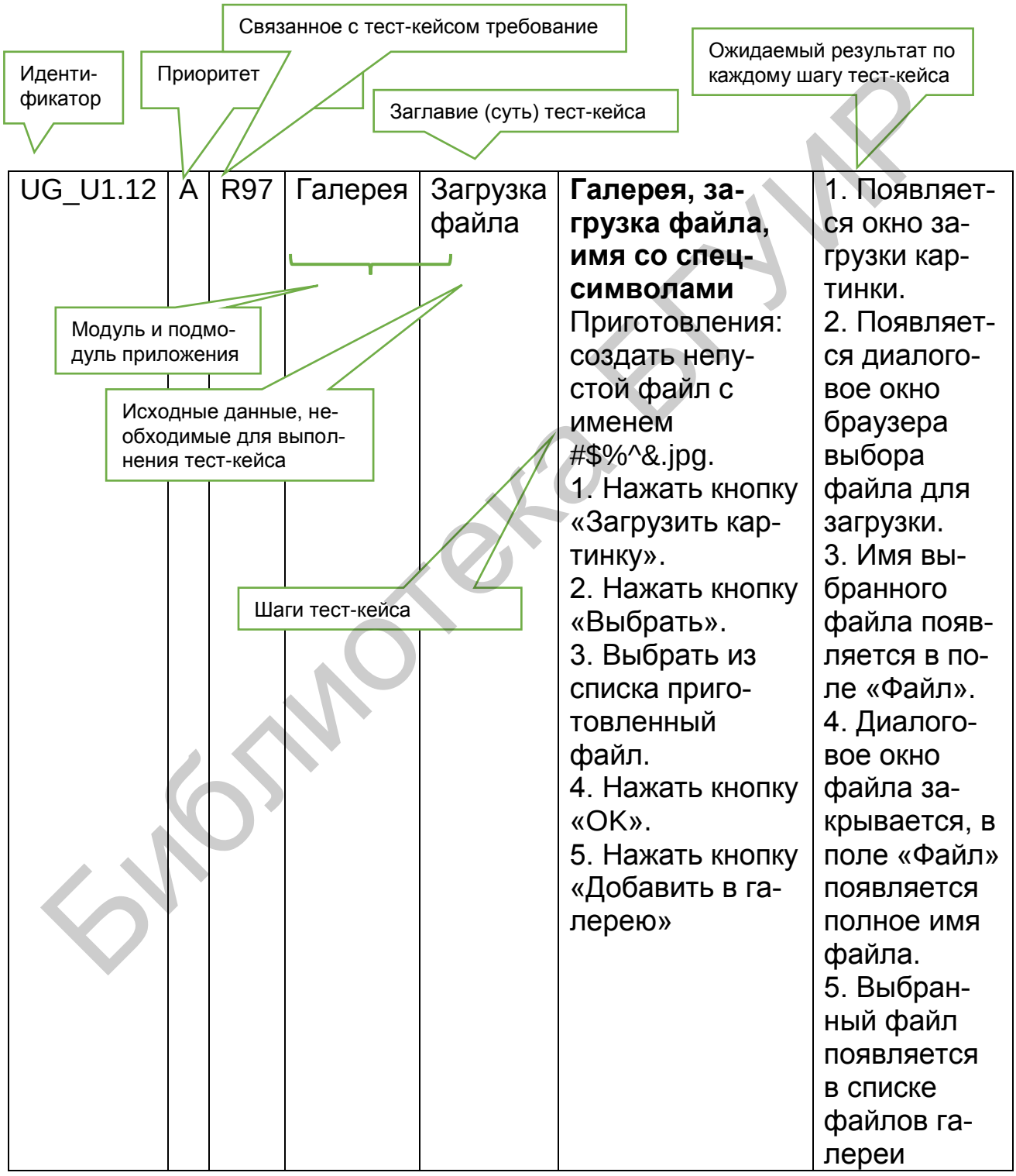
Как уже было сказано выше, термин «тест-кейс» может относиться к формальной записи тест-кейса в виде технического документа. Эта за-

[73] **Coverage (test coverage).** The degree, expressed as a percentage, to which a specified coverage item (an entity or property used as a basis for test coverage, e.g. equivalence partitions or code statements) has been exercised by a test suite. ISTQB Glossary.

пись имеет общепринятую структуру, компоненты которой называются атрибутами (полями) тест-кейса.

В зависимости от инструмента управления тест-кейсом внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

Общий вид всей структуры тест-кейса представлен на рисунке 2.а.



UG_U1.12	A	R97	Галерея	Загрузка файла	**Галерея, загрузка файла, имя со спецсимволами** Приготовления: создать непустой файл с именем #$%^&.jpg. 1. Нажать кнопку «Загрузить картинку». 2. Нажать кнопку «Выбрать». 3. Выбрать из списка приготовленный файл. 4. Нажать кнопку «OK». 5. Нажать кнопку «Добавить в галерею»	1. Появляется окно загрузки картинки. 2. Появляется диалоговое окно браузера выбора файла для загрузки. 3. Имя выбранного файла появляется в поле «Файл». 4. Диалоговое окно файла закрывается, в поле «Файл» появляется полное имя файла. 5. Выбранный файл появляется в списке файлов галереи

Рисунок 2.а – Общий вид тест-кейса

Теперь рассмотрим каждый атрибут подробно.

Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один тест-кейс от другого и используемое во всевозможных ссылках. В общем случае идентификатор тест-кейса может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления тест-кейсами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить цель тест-кейса и часть приложения (или требований), к которой он относится.

Приоритет (priority) показывает важность тест-кейса. Он может быть выражен буквами (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («крайне высокий», «высокий», «средний», «низкий», «крайне низкий») или иным удобным способом. Количество градаций также не фиксировано, но чаще всего лежит в диапазоне от трёх до пяти.

Приоритет тест-кейса может коррелировать с:

- Важностью требования, пользовательского сценария (см. «Пользовательские сценарии (сценарии использования)» в подразделе 2.5) или функции, с которыми связан тест-кейс.
- Потенциальной важностью дефекта (см. «Важность (severity)» в подразделе 2.5), на поиск которого направлен тест-кейс.
- Степенью риска, связанного с проверяемым тест-кейсом требованием, сценарием или функцией.

Основная задача этого атрибута – упрощение распределения внимания и усилий команды (более высокоприоритетные тест-кейсы получают их больше), а также упрощение планирования и принятия решения о том, чем можно пожертвовать в некоей форс-мажорной ситуации, не позволяющей выполнить все запланированные тест-кейсы.

Связанное с тест-кейсом требование (requirement) показывает то основное требование, проверке выполнения которого посвящён тест-кейс (основное – потому, что один тест-кейс может затрагивать несколько требований). Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость (см. «Прослеживаемость» в подразделе 2.5).

Частые вопросы, связанные с заполнением этого поля, таковы:

- Можно ли его оставить пустым? Да. Тест-кейс вполне мог разрабатываться вне прямой привязки к требованиям, и (пока) значение этого поля определить сложно. Хоть такой вариант и не считается хорошим, он достаточно распространён.
- Можно ли в этом поле указывать несколько требований? Да, но чаще всего стараются выбрать одно самое главное или «более высокоуровневое» (например, вместо того, чтобы перечислять R56.1, R56.2, R56.3 и т. д., можно просто написать R56). Чаще всего в инструментах управления тестами это поле представляет со-

бой выпадающий список, где можно выбрать только одно значение, и этот вопрос становится неактуальным. К тому же многие тест-кейсы всё же направлены на проверку строго одного требования, и для них этот вопрос также неактуален.

Модуль и подмодуль приложения (module and submodule) указывают на части приложения, к которым относится тест-кейс, и позволяют лучше понять его цель.

Идея деления приложения на модули и подмодули проистекает из того, что в сложных системах практически невозможно охватить взглядом весь проект целиком, и вопрос «как протестировать это приложение» становится недопустимо сложным. Тогда приложение логически разделяется на компоненты (модули), а те, в свою очередь, – на более мелкие компоненты (подмодули). И вот уже для таких небольших частей приложения придумать чек-листы и создать хорошие тест-кейсы становится намного проще.

Как правило, иерархия модулей и подмодулей создаётся как единый набор для всей проектной команды, чтобы исключить путаницу из-за того, что разные люди будут использовать разные подходы к такому разделению или даже просто разные названия одних и тех же частей приложения.

Теперь – самое сложное: как выбираются модули и подмодули. В реальности проще всего отталкиваться от архитектуры и дизайна приложения. Например, в уже знакомом нам приложении можно выделить такую иерархию модулей и подмодулей:

- Механизм запуска:
  - механизм анализа параметров;
  - механизм сборки приложения;
  - механизм обработки ошибочных ситуаций.
- Механизм взаимодействия с файловой системой:
  - механизм обхода дерева SOURCE_DIR;
  - механизм обработки ошибочных ситуаций.
- Механизм преобразования файлов:
  - механизм определения кодировок;
  - механизм преобразования кодировок;
  - механизм обработки ошибочных ситуаций.
- Механизм ведения журнала:
  - механизм записи журнала;
  - механизм отображения журнала в консоли;
  - механизм обработки ошибочных ситуаций.

Согласитесь, что такие длинные названия с постоянно повторяющимся словом «механизм» читать и запоминать сложно. Перепишем:

- Стартер:
  - анализатор параметров;

- сборщик приложения;
- обработчик ошибок.

- Сканер:
  - обходчик;
  - обработчик ошибок.
- Преобразователь:
  - детектор;
  - конвертер;
  - обработчик ошибок.
- Регистратор:
  - дисковый регистратор;
  - консольный регистратор;
  - обработчик ошибок.

Но что делать, если мы не знаем «внутренностей» приложения (или не очень разбираемся в программировании)? Модули и подмодули можно выделять на основе графического интерфейса пользователя (крупные области и элементы внутри них), на основе решаемых приложением задач и подзадач и т. д. Главное, чтобы эта логика была одинаковым образом применена ко всему приложению.

Внимание! Частая ошибка! Модуль и подмодуль приложения – это НЕ действия, это именно структурные части, «куски» приложения. В заблуждение вас могут ввести такие названия, как, например, «печать, настройка принтера» (но здесь имеются в виду именно части приложения, отвечающие за печать и за настройку принтера (и названы они отглагольными существительными), а не процесс печати или настройки принтера).

Сравните (на примере человека): «дыхательная система, лёгкие» – это модуль и подмодуль, а «дыхание», «сопение», «чихание» – нет; «голова, мозг» – это модуль и подмодуль, а «кивание», «думание» – нет.

Наличие полей «Модуль и подмодуль приложения» улучшает такое свойство тест-кейса, как прослеживаемость (см. «Прослеживаемость» в подразделе 2.5).

Заглавие (суть) тест-кейса (title) призвано упростить быстрое понимание основной идеи тест-кейса без обращения к его остальным атрибутам. Именно это поле является наиболее информативным при просмотре списка тест-кейсов.

Сравните (таблица 2.с).

Таблица 2.с – Сравнение тестов

Плохо	Хорошо
Тест 1	Запуск, одна копия, верные параметры
Тест 2	Запуск одной копии с неверными путями
Тест 78 (улучшенный)	Запуск, много копий, без конфликтов
Остановка	Остановка по Ctrl+C
Закрытие	Остановка закрытием консоли
…	…

Заглавие тест-кейса может быть полноценным предложением, фразой, набором словосочетаний – главное, чтобы выполнялись следующие условия:

- Информативность.
- Хотя бы относительная уникальность (чтобы не путать разные тест-кейсы).

Внимание! Частая ошибка! Если инструмент управления тест-кейсами не требует писать заглавие, его **всё равно надо писать**. Тест-кейсы без заглавий превращаются в запутанную информацию, использование которой сопряжено с колоссальными и совершенно бессмысленными затратами.

И ещё одна небольшая мысль, которая может помочь лучше формулировать заглавия. В дословном переводе с английского «test case» обозначает «тестовый случай (ситуация)». Так вот, заглавие как раз и описывает этот случай (ситуацию), т. е. что происходит в тест-кейсе, какую ситуацию он проверяет.

Исходные данные, необходимые для выполнения тест-кейса (precondition, preparation, initial data, setup), позволяют описать всё то, что должно быть подготовлено до начала выполнения тест-кейса, например:

- Состояние базы данных.
- Состояние файловой системы и её объектов.
- Состояние серверов и сетевой инфраструктуры.

ОЧЕНЬ ВАЖНО! Всё, что описывается в этом поле, готовится БЕЗ использования тестируемого приложения, и, таким образом, если здесь возникают проблемы, нельзя писать отчёт о дефекте в приложении. Эта мысль очень и очень важна, потому поясним её простым жизненным примером. Представьте, что вы дегустируете конфеты. В поле «исходные данные» можно прописать «купить конфеты таких-то сортов в таком-то количестве». Если таких конфет нет в продаже, если закрыт магазин, если не хватило денег и т. д. – всё это НЕ проблемы вкуса конфет, и нельзя писать отчёт о дефекте конфет вида «конфеты невкусные потому, что закрыт магазин».

Шаги тест-кейса (steps) описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса. Общие рекомендации по написанию шагов таковы:

- Начинайте с понятного и очевидного места, не пишите лишних начальных шагов (запуск приложения, очевидные операции с интерфейсом и т. п.).
- Даже если в тест-кейсе всего один шаг, нумеруйте его (иначе возрастает вероятность в будущем случайно «приклеить» описание этого шага к новому тексту).
- Если вы пишете на русском языке, используйте безличную форму (например, «открыть», «ввести», «добавить» вместо «откройте», «введите», «добавьте»).
- Соотносите степень детализации шагов и их параметров с целью тест-кейса, его сложностью, уровнем и т. д. – в зависимости от этих и многих других факторов степень детализации может варьироваться от общих идей до предельно чётко прописанных значений и указаний.
- Ссылайтесь на предыдущие шаги и их диапазоны для сокращения объёма текста (например, «повторить шаги 3–5 со значением…»).
- Пишите шаги последовательно, без условных конструкций вида «если…, то…».

Внимание! Частая ошибка! Категорически запрещено ссылаться на шаги из других тест-кейсов и другие тест-кейсы целиком: если те, другие, тест-кейсы будут изменены или удалены, ваш тест-кейс начнёт ссылаться на неверные данные или в пустоту, а если в процессе выполнения те, другие, тест-кейсы или шаги приведут к возникновению ошибки, вы не сможете закончить выполнение вашего тест-кейса.

Ожидаемые результаты (expected results) по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.

По написанию ожидаемых результатов можно порекомендовать следующее:

- Описывайте поведение системы так, чтобы исключить субъективное толкование (например, «приложение работает верно» – плохо, «появляется окно с надписью…» – хорошо).
- Пишите ожидаемый результат по всем шагам без исключения, если у вас есть хоть малейшие сомнения в том, что результат некоего шага будет совершенно тривиальным и очевидным (если вы всё же пропускаете ожидаемый результат для какого-то триви-

ального действия, лучше оставить в списке ожидаемых результатов пустую строку – это облегчает восприятие).

- Пишите кратко, но не в ущерб информативности.
- Избегайте условных конструкций вида «если…, то…».

Внимание! Частая ошибка! В ожидаемых результатах ВСЕГДА описывается КОРРЕКТНАЯ работа приложения. Нет и не может быть ожидаемого результата в виде «приложение вызывает ошибку в операционной системе и аварийно завершается с потерей всех пользовательских данных».

При этом корректная работа приложения вполне может предполагать отображение сообщений о неверных действиях пользователя или неких критических ситуациях. Так, сообщение «Невозможно сохранить файл по указанному пути: на целевом носителе недостаточно свободного места» – это не ошибка приложения, это его совершенно нормальная и правильная работа. Ошибкой приложения (в этой же ситуации) было бы отсутствие такого сообщения, и/или повреждение, или потеря записываемых данных.

2.4 Свойства качественных тест-кейсов

Даже правильно оформленный тест-кейс может оказаться некачественным, если в нём нарушено одно из следующих свойств.

Правильный технический язык. Это свойство в равной мере относится и к требованиям, и к тест-кейсам, и к отчётам о дефектах – к любой документации. Из самого общего и важного напомним и добавим:

- Пишите лаконично, но понятно.
- Используйте безличную форму глаголов (например, «открыть» вместо «откройте»).
- Обязательно указывайте точные имена и технически верные названия элементов приложения.
- Не объясняйте базовые принципы работы с компьютером (предполагается, что ваши коллеги знают, что такое, например, «пункт меню» и как с ним работать).

Баланс между специфичностью и общностью. Тест-кейс считается тем более специфичным, чем более детально в нём расписаны конкретные действия, конкретные значения и т. д., т. е. чем в нём больше конкретики. Соответственно, тест-кейс считается тем более общим, чем в нём меньше конкретики.

Рассмотрим поля «шаги» и «ожидаемые результаты» двух тест-кейсов (подумайте, какой тест-кейс вы бы посчитали хорошим, а какой – плохим и почему).

Тест-кейс 1 представлен в таблице 2.d.

Таблица 2.d – Тест-кейс 1

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок Приготовления: • Создать папки C:/A, C:/B, C:/C, C:/D. • Разместить в папке C:/D файлы 1.html, 2.txt, 3.md из прилагаемого архива. 1. Запустить приложение, выполнив команду «php converter.php c:/a c:/b c:/c/converter.log». 2. Скопировать файлы 1.html, 2.txt, 3.md из папки C:/D в папку C:/A. 3. Остановить приложение нажатием Crtl+C	1. Отображается консольный журнал приложения с сообщением «текущее_время started, source dir c:/a, destination dir c:/b, log file c:/c/converter.log», в папке C:/C появляется файл converter.log, в котором появляется запись «текущее_время started, source dir c:/a, destination dir c:/b, log file c:/c/converter.log». 2. Файлы 1.html, 2.txt, 3.md появляются в папке C:/A, затем пропадают оттуда и появляются в папке C:/B. В консольном журнале и файле C:/C/converter.log появляются сообщения (записи) «текущее_время processing 1.html (KOI8-R)», «текущее_время processing 2.txt (CP-1251)», «текущее_время processing 3.md (CP-866)». 3. В файле C:/C/converter.log появляется запись «текущее_время closed». Приложение завершает работу

Тест-кейс 2 представлен в таблице 2.е.

Таблица 2.е – Тест-кейс 2

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок 1. Выполнить конвертацию трёх файлов допустимого размера в трёх разных кодировках всех трёх допустимых форматов	1. Файлы перемещаются в папку-приёмник, кодировка всех файлов меняется на UTF-8

Если вернуться к вопросу «какой тест-кейс вы бы посчитали хорошим, а какой – плохим и почему», то ответ таков: оба тест-кейса плохие потому, что первый является слишком специфичным, а второй – слишком общим. Можно сказать, что здесь до абсурда доведены идеи низкоуровневых (см. «Низкоуровневый тест-кейс» в подразделе 2.5) и высокоуровневых (см. «Высокоуровневый тест-кейс» в подразделе 2.5) тест-кейсов.

Почему плоха излишняя специфичность (тест-кейс 1):

- При повторных выполнениях тест-кейса всегда будут выполняться строго одни и те же действия со строго одними и теми же данными, что снижает вероятность обнаружения ошибки.
- Возрастает время написания, доработки и даже просто прочтения тест-кейса.
- В случае выполнения тривиальных действий опытные специалисты тратят дополнительные мыслительные ресурсы в попытках понять, что же они упустили из виду, т. к. они привыкли, что так описываются только самые сложные и неочевидные ситуации.

Почему плоха излишняя общность (тест-кейс 2):

- Тест-кейс сложен для выполнения начинающими тестировщиками или даже опытными специалистами, лишь недавно подключившимися к проекту.
- Недобросовестные сотрудники склонны халатно относиться к таким тест-кейсам.
- Тестировщик, выполняющий тест-кейс, может понять его иначе, чем было задумано автором (и в итоге будет выполнен фактически совсем другой тест-кейс).

Выход из этой ситуации состоит в том, чтобы придерживаться золотой середины (хотя, конечно же, какие-то тесты будут чуть более специфичными, какие-то – чуть более общими). Вот пример такого срединного подхода.

Тест-кейс 3 представлен в таблице 2.f.

Таблица 2.f – Тест-кейс 3

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок Приготовления: • Создать в корне любого диска четыре отдельные папки для входных файлов, выходных файлов, файла журнала и временного хранения тестовых файлов. • Распаковать содержимое прилагаемого архива в папку для временного хранения тестовых файлов. 1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала – произвольное). 2. Скопировать файлы из папки для временного хранения в папку для входных файлов. 3. Остановить приложение	1. Приложение запускается и выводит сообщение о своём запуске в консоль и файл журнала. 2. Файлы из папки для входных файлов перемещаются в папку для выходных файлов, в консоли и файле журнала отображаются сообщения о конвертации каждого из файлов с указанием его исходной кодировки. 3. Приложение выводит сообщение о завершении работы в файл журнала и завершает работу

В этом тест-кейсе есть всё необходимое для понимания и выполнения, но при этом он стал короче и проще для выполнения, а отсутствие строго указанных значений приводит к тому, что при многократном выполнении тест-кейса (особенно – разными тестировщиками) конкретные параметры будут менять свои значения, что увеличивает вероятность обнаружения ошибки.

Ещё раз главная мысль: сами по себе специфичность или общность тест-кейса не являются чем-то плохим, но резкий перекос в ту или иную сторону снижает качество тест-кейса.

Баланс между простотой и сложностью. Здесь не существует академических определений, но принято считать, что простой тест-кейс оперирует одним объектом (или в нём явно виден главный объект), а также содержит небольшое количество тривиальных действий; сложный тест-кейс оперирует несколькими равноправными объектами и содержит много нетривиальных действий.

Преимущества простых тест-кейсов:

- Их можно быстро прочесть, легко понять и выполнить.
- Они понятны начинающим тестировщикам и новым людям на проекте.
- Они делают наличие ошибки очевидным (как правило, в них предполагается выполнение повседневных тривиальных действий, проблемы с которыми видны невооружённым взглядом и не вызывают дискуссий).
- Они упрощают начальную диагностику ошибки, т. к. сужают круг поиска.

Преимущества сложных тест-кейсов:

- При взаимодействии многих объектов повышается вероятность возникновения ошибки.
- Пользователи, как правило, используют сложные сценарии, а потому сложные тесты более полноценно эмулируют работу пользователей.
- Программисты редко проверяют такие сложные случаи (и они совершенно не обязаны это делать).

Рассмотрим примеры.

Слишком простой тест-кейс представлен в таблице 2.g.

Таблица 2.g – Простой тест-кейс

Шаги	Ожидаемые результаты
Запуск приложения 1. Запустить приложение	1. Приложение запускается

Слишком сложный тест-кейс представлен в таблице 2.h.

Таблица 2.h – Сложный тест-кейс

Шаги	Ожидаемые результаты
Повторная конвертация Приготовления: • Создать в корне любого диска три отдельные папки для входных файлов, выходных файлов, файла журнала. • Подготовить набор из нескольких файлов максимального поддерживаемого размера поддерживаемых форматов с поддерживаемыми кодировками, а также нескольких файлов допустимого размера, но недопустимого формата.	2. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов.

Продолжение таблицы 2.h

Шаги	Ожидаемые результаты
1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала – произвольное). 2. Скопировать в папку для входных файлов несколько файлов допустимого формата. 3. Переместить сконвертированные файлы из папки с результирующими файлами в папку для входных файлов. 4. Переместить сконвертированные файлы из папки с результирующими файлами в папку с набором файлов для теста. 5. Переместить все файлы из папки с набором файлов для теста в папку для входных файлов. 6. Переместить сконвертированные файлы из папки с результирующими файлами в папку для входных файлов	3. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов. 5. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов допустимого формата и сообщения об игнорировании файлов недопустимого формата. 6. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов допустимого формата и сообщения об игнорировании файлов недопустимого формата

Этот тест-кейс одновременно является слишком сложным по избыточности действий и по спецификации лишних данных и операций.

Задание 2.c: перепишите этот тест-кейс, устранив его недостатки, но сохранив общую цель (проверку повторной конвертации уже ранее сконвертированных файлов).

Примером хорошего простого тест-кейса может служить тест-кейс 3.

Пример хорошего сложного тест-кейса может выглядеть так, как показано в таблице 2.i.

Таблица 2.i – Хороший сложный тест-кейс

Шаги	Ожидаемые результаты
Много копий приложения, конфликт файловых операций Приготовления: • Создать в корне любого диска три отдельные папки для входных файлов, выходных файлов, файла журнала. • Подготовить набор из нескольких файлов максимального поддерживаемого размера поддерживаемых форматов с поддерживаемыми кодировками. 1. Запустить первую копию приложения, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала – произвольное). 2. Запустить вторую копию приложения с теми же параметрами (см. шаг 1). 3. Запустить третью копию приложения с теми же параметрами (см. шаг 1). 4. Изменить приоритет процессов второй (“high”) и третей (“low”) копий. 5. Скопировать подготовленный набор исходных файлов в папку для входных файлов	3. Все три копии приложения запускаются, в файле журнала появляются последовательно три записи о запуске приложения. 5. Файлы постепенно перемещаются из входной в выходную папку, в консоли и файле журнала появляются сообщения об успешной конвертации файлов, а также (возможно) сообщения вида: a. “source file inaccessible, retrying”. b. “destination file inaccessible, retrying”. c. “log file inaccessible, retrying”. Ключевым показателем корректной работы является успешная конвертация всех файлов, а также появление в консоли и файле журнала сообщений об успешной конвертации каждого файла (от одной до трёх записей на каждый файл). Сообщения (предупреждения) о недоступности входного файла, выходного файла или файла журнала также являются показателем корректной работы приложения, однако их количество зависит от многих внешних факторов и не может быть спрогнозировано заранее

Иногда более сложные тест-кейсы являются также и более специфичными, но это лишь общая тенденция, а не закон. Также нельзя по сложности тест-кейса однозначно судить о его приоритете (в нашем примере хорошего сложного тест-кейса он явно будет иметь очень низкий приоритет, т. к. проверяемая им ситуация является искусственной и крайне маловероятной, но бывают и сложные тесты с самым высоким приоритетом).

Как и в случае специфичности и общности, сами по себе простота или сложность тест-кейсов не являются чем-то плохим (более того – рекомендуется начинать разработку и выполнение тест-кейсов с простых, а затем переходить ко всё более и более сложным), однако излишняя простота и излишняя сложность также снижают качество тест-кейса.

«Показательность» (высокая вероятность обнаружения ошибки). Начиная с уровня тестирования критического пути можно утверждать, что тест-кейс является тем более хорошим, чем он более показателен (с большей вероятностью обнаруживает ошибку). Именно поэтому мы считаем непригодными слишком простые тест-кейсы – они непоказательны.

Пример непоказательного (плохого) тест-кейса показан в таблице 2.j.

Таблица 2.j – Плохой тест-кейс

Шаги	Ожидаемые результаты
Запуск и остановка приложения 1. Запустить приложение с корректными параметрами. 2. Завершить работу приложения	1. Приложение запускается. 2. Приложение завершает работу

Пример показательного (хорошего) тест-кейса показан в таблице 2.k.

Таблица 2.k – Плохой тест-кейс

Шаги	Ожидаемые результаты
Запуск с некорректными параметрами, несуществующие пути 1. Запустить приложение со всеми тремя параметрами (SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME), значения которых указывают на несуществующие в файловой системе пути (например: z:\src\, z:\dst\, z:\log.txt при условии, что в системе нет логического диска z)	1. В консоли отображаются нижеуказанные сообщения, приложение завершает работу. Сообщения: a. SOURCE_DIR: directory not exists or inaccessible. b. DESTINATION_DIR: directory not exists or inaccessible. c. LOG_FILE_NAME: wrong file name or inaccessible path

Обратите внимание, что показательный тест-кейс по-прежнему остался достаточно простым, но он проверяет ситуацию, возникновение ошибки в которой несравненно более вероятно, чем в ситуации, описываемой плохим непоказательным тест-кейсом.

Также можно сказать, что показательные тест-кейсы часто выполняют какие-то «интересные действия», т. е. такие действия, которые едва ли будут выполнены просто в процессе работы с приложением (например: «сохранить файл» – это обычное тривиальное действие, которое явно будет выполнено не одну сотню раз даже самими разработчиками, а вот «сохранить файл на носитель, защищённый от записи», «сохранить файл на носитель с недостаточным объёмом свободного пространства», «сохранить файл в папку, к которой нет доступа» – это уже гораздо более интересные и нетривиальные действия).

Последовательность в достижении цели. Суть этого свойства выражается в том, что все действия в тест-кейсе направлены на следование единой логике и достижение единой цели и не содержат никаких отклонений.

Примерами правильной реализации этого свойства могут служить представленные в этом подразделе в избытке примеры хороших тест-кейсов. А нарушение может выглядеть так, как показано в таблице 2.l.

Таблица 2.l – Ошибки в тест-кейсах

Шаги	Ожидаемые результаты
Конвертация из всех поддерживаемых кодировок Приготовления: • Создать в корне любого диска четыре отдельные папки для входных файлов, выходных файлов, файла журнала и временного хранения тестовых файлов. • Распаковать содержимое прилагаемого архива в папку для временного хранения тестовых файлов. 1. Запустить приложение, указав в параметрах соответствующие пути из приготовления к тесту (имя файла журнала – произвольное).	1. Приложение запускается и выводит сообщение о своём запуске в консоль и файл журнала.

Продолжение таблицы 2.l

Шаги	Ожидаемые результаты
2. Скопировать файлы из папки для временного хранения в папку для входных файлов. *3. Остановить приложение.* *4. Удалить файл журнала.* *5. Повторно запустить приложение с теми же параметрами.* 6. Остановить приложение	2. Файлы из папки для входных файлов перемещаются в папку для выходных файлов, в консоли и файле журнала отображаются сообщения о конвертации каждого из файлов с указанием его исходной кодировки. *3. Приложение выводит сообщение о завершении работы в файл журнала и завершает работу.* *5. Приложение запускается и выводит сообщение о своём запуске в консоль и заново созданный файл журнала.* 6. Приложение выводит сообщение о завершении работы в файл журнала и завершает работу

Шаги 3–5 никак не соответствуют цели тест-кейса, состоящей в проверке корректности конвертации входных данных, представленных во всех поддерживаемых кодировках.

Отсутствие лишних действий. Чаще всего это свойство подразумевает, что не нужно в шагах тест-кейса долго и по пунктам расписывать то, что можно заменить одной фразой (таблица 2.m).

Таблица 2.m – Лишние действия в тест-кейсах

Плохо	Хорошо
1. Указать в качестве первого параметра приложения путь к папке с исходными файлами. 2. Указать в качестве второго параметра приложения путь к папке с конечными файлами. 3. Указать в качестве третьего параметра приложения путь к файлу журнала. 4. Запустить приложение	1. Запустить приложение со всеми тремя корректными параметрами (например: c:\src\, c:\dst\, c:\log.txt – при условии, что соответствующие папки существуют и доступны приложению)

Вторая по частоте ошибка – начало каждого тест-кейса с запуска приложения и подробного описания по приведению его в то или иное состояние. В наших примерах мы рассматриваем каждый тест-кейс как существующий в единственном виде в изолированной среде и потому вынуждены осознанно допускать эту ошибку (иначе тест-кейс будет неполным), но в реальной жизни на запуск приложения будут свои тесты, а длинный путь из многих действий можно описать как одно действие, из контекста которого понятно, как это действие выполнить. Следующий пример тест-кейса не относится к нашему «Конвертеру файлов», но очень хорошо иллюстрирует эту мысль (таблица 2.n).

Таблица 2.n – Частые ошибки в тест-кейсах

Плохо	**Хорошо**
1. Запустить приложение. 2. Выбрать в меню пункт «Файл». 3. Выбрать подпункт «Открыть». 4. Перейти в папку, в которой находится хотя бы один файл формата DOCX с тремя и более страницами	1. Открыть DOCX-файл с тремя и более страницами

И сюда же можно отнести ошибку с повторением одних и тех же приготовлений во множестве тест-кейсов (да, по описанным выше причинам в примерах мы снова вынужденно делаем так, как на практике делать не надо). Куда удобнее объединить тесты в набор и указать приготовления один раз, подчеркнув, нужно или нет их выполнять перед каждым тест-кейсом в наборе.

Проблема с подготовительными (и финальными) действиями идеально решена в автоматизированном модульном тестировании[74] с использованием фреймворков наподобие JUnit или TestNG – там существует специальный «механизм фиксаций» (fixture), автоматически выполняющий указанные действия перед каждым отдельным тестовым методом или после него.

Неизбыточность по отношению к другим тест-кейсам. В процессе создания множества тест-кейсов очень легко оказаться в ситуации, когда два и более тест-кейса фактически выполняют одни и те же проверки, преследуют одни и те же цели, направлены на поиск одних и тех же проблем. Существуют разные способы минимизации количества таких тест-кейсов, такие как использование классов эквивалентности и граничных условий.

[74] **Unit testing (component testing).** The testing of individual software components. ISTQB Glossary.

Если вы обнаруживаете несколько тест-кейсов, дублирующих задачи друг друга, лучше всего или удалить все, кроме одного, самого показательного, или перед удалением остальных на их основе доработать этот выбранный самый показательный тест-кейс.

Демонстративность (способность демонстрировать обнаруженную ошибку очевидным образом). Ожидаемые результаты должны быть подобраны и сформулированы таким образом, чтобы любое отклонение от них сразу же бросалось в глаза и становилось очевидным, что произошла ошибка. Сравните выдержки из двух тест-кейсов.

Выдержка из недемонстративного тест-кейса представлена в таблице 2.о.

Таблица 2.о – Недемонстративный тест-кейс

Шаги	Ожидаемые результаты
5. Разместить в файле текст «Пример длинного текста, содержащего символы русского и английского алфавита вперемешку.» в кодировке KOI8-R (в слове «Пример» буквы «р» – английские). 6. Сохранить файл под именем «test. txt» и отправить файл на конвертацию. 7. Переименовать файл в «test.txt»	6. Приложение игнорирует файл. 7. Текст принимает корректный вид в кодировке UTF-8 с учётом английских букв

Выдержка из демонстративного тест-кейса представлена в таблице 2.р.

Таблица 2.р – Демонстративный тест-кейс

Шаги	Ожидаемые результаты
5. Разместить в файле текст «╨Я╤А╨╕╨╝╨╡╤А ╤В╨╡╨║╤Б╤В╨░.» (Эти символы представляют собой словосочетание «Пример текста.» в кодировке KOI8-R). 6. Отправить файл на конвертацию	6. Текст принимает вид: «Пример текста.» (кодировка UTF8)

В первом случае тест-кейс не подходит не только из-за неясности формулировки «корректный вид в кодировке UTF-8 с учётом английских букв», там также очень легко допустить ошибки при выполнении:

- Забыть сконвертировать вручную входной текст в KOI8-R.
- Не заметить, что в первый раз расширение начинается с пробела.

- Забыть заменить в слове «Пример» букву «р» на английскую.
- Из-за расплывчатости формулировки ожидаемого результата принять ошибочное, но выглядящее правдоподобно поведение за верное.

Второй тест-кейс чётко ориентирован на свою цель по проверке конвертации (не содержит странной проверки с игнорированием файла с неверным расширением) и описан так, что его выполнение не представляет никаких сложностей, а любое отклонение фактического результата от ожидаемого будет сразу же заметно.

Прослеживаемость. Из содержащейся в качественном тест-кейсе информации должно быть понятно, какую часть приложения, какие функции и какие требования он проверяет. Частично это свойство достигается через заполнение соответствующих полей тест-кейса («Ссылка на требование», «Модуль», «Подмодуль»), но и сама логика тест-кейса играет не последнюю роль, т. к. в случае серьёзных нарушений этого свойства можно долго с удивлением смотреть, например, на какое требование ссылается тест-кейс, и пытаться понять, как же они друг с другом связаны.

Пример непрослеживаемого тест-кейса представлен в таблице 2.q.

Таблица 2.q – Непрослеживаемый тест-кейс

Требование	**Модуль**	**Подмодуль**	**Шаги**	**Ожидаемые результаты**
ПТ-4	Приложение		**Совмещение кодировок** Приготовления: файл с несколькими допустимыми и недопустимыми кодировками**.** 1. Передать файл на конвертацию	1. Допустимые кодировки конвертируются верно, недопустимые остаются без изменений

Да, этот тест-кейс не подходит сам по себе (в качественном тест-кейсе сложно получить ситуацию непрослеживаемости), но в нём есть и особые недостатки, затрудняющие прослеживаемость:

- Ссылка на несуществующее требование (убедитесь сами, требования ПТ-4 нет).
- В поле «Модуль» указано значение «Приложение» (по большому счёту можно было оставлять это поле пустым – это было бы столь же информативно), поле «Подмодуль» не заполнено.

- По заглавию и шагам можно предположить, что этот тест-кейс ближе всего к ДС-5.1 и ДС-5.3, но сформулированный ожидаемый результат не следует явно из этих требований.

Пример прослеживаемого тест-кейса представлен в таблице 2.r.

Таблица 2.r – Прослеживаемый тест-кейс

Требование	**Модуль**	**Подмодуль**	**Шаги**	**Ожидаемые результаты**
ДС-2.4, ДС-3.2	Стартер	Обработчик ошибок	**Запуск с некорректными параметрами, несуществующие пути** 1. Запустить приложение со всеми тремя параметрами, значения которых указывают на несуществующие в файловой системе пути	1. В консоли отображаются нижеуказанные сообщения, приложение завершает работу. Сообщения a. SOURCE_DIR: directory not exists or inaccessible. b. DESTINTION_DIR: directory not exists or inaccessible. c. LOG_FILE_NAME: wrong file name or inaccessible path

Можно подумать, что этот тест-кейс затрагивает ДС-2 и ДС-3 целиком, но в поле «Требование» есть вполне чёткая конкретизация, к тому же указанные модуль, подмодуль и сама логика тест-кейса устраняют оставшиеся сомнения.

Возможность повторного использования. Это свойство редко выполняется для низкоуровневых тест-кейсов (см. «Низкоуровневый тест-кейс» в подразделе 2.5), но при создании высокоуровневых тест-кейсов (см. «Высокоуровневый тест-кейс» в подразделе 2.5) можно добиться таких формулировок, при которых тест-кейс практически без изменений можно будет использовать для тестирования аналогичной функциональности в других проектах или других областях приложения.

Примером тест-кейса, который тяжело использовать повторно, может являться практически любой тест-кейс с высокой специфичностью.

Не самым идеальным, но очень наглядным примером тест-кейса, который может быть легко использован в разных проектах, может служить следующий тест-кейс (таблица 2.s).

Таблица 2.s – Наглядный тест-кейс

Шаги	Ожидаемые результаты
Запуск, все параметры некорректны 1. Запустить приложение, указав в качестве всех параметров заведомо некорректные значения	1. Приложение запускается, после чего выводит сообщение с описанием сути проблемы с каждым из параметров и завершает работу

Соответствие принятым шаблонам оформления и традициям. С шаблонами оформления, как правило, проблем не возникает: они строго определены имеющимся образцом или вообще экранной формой инструментального средства управления тест-кейсами. Что же касается традиций, то они отличаются даже в разных командах в рамках одной компании, и тут невозможно дать иного совета, кроме как «почитайте уже готовые тест-кейсы перед тем, как писать свои».

2.5 Наборы тест-кейсов

Терминология и общие сведения

Набор тест-кейсов (test case suite[75], test suite, test set) – совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего тест-кейса.

Внимание! Из-за особенностей перевода очень часто вместо «набор текст-кейсов» говорят «тестовый сценарий». Формально это можно считать ошибкой, но это явление приобрело настолько широкое распространение, что стало вариантом нормы.

Как мы только что убедились на примере множества отдельных тест-кейсов, крайне неудобно (более того, это ошибка!) каждый раз писать в каждом тест-кейсе одни и те же приготовления и повторять одни и те же начальные шаги.

Намного удобнее объединить несколько тест-кейсов в набор или последовательность. И здесь мы приходим к классификации наборов тест-кейсов.

В общем случае наборы тест-кейсов можно разделить на свободные (порядок выполнения тест-кейсов не важен) и последовательные (порядок выполнения тест-кейсов важен).

[75] **Test case suite (test suite, test set).** A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one. ISTQB Glossary.

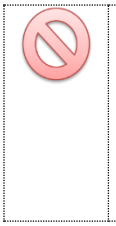
Преимущества свободных наборов:

- Тест-кейсы можно выполнять в любом удобном порядке, а также создавать «наборы внутри наборов».
- Если какой-то тест-кейс завершился ошибкой, это не повлияет на возможность выполнения других тест-кейсов.

Преимущества последовательных наборов:

- Каждый следующий в наборе тест-кейс в качестве входного состояния приложения получает результат работы предыдущего тест-кейса, что позволяет сильно сократить количество шагов в отдельных тест-кейсах.
- Длинные последовательности действий куда лучше имитируют работу реальных пользователей, чем отдельные воздействия на приложение.

Пользовательские сценарии (сценарии использования)



В данном случае речь НЕ идёт о use cases (вариантах использования), представляющих собой форму требований (см. подраздел 1.4). Пользовательские сценарии как техника тестирования куда менее формализованы, хотя и могут строиться на основе вариантов использования.

К отдельному подвиду последовательных наборов тест-кейсов (или даже неоформленных идей тест-кейсов, таких, как пункты чек-листа) можно отнести пользовательские сценарии[76] (или сценарии использования), представляющие собой цепочки действий, выполняемых пользователем в определённой ситуации для достижения определённой цели.

Поясним это сначала на примере, не относящемся к «Конвертеру файлов». Допустим, пользователь хочет распечатать табличку на дверь кабинета с текстом «Идёт работа, не стучать!» Для этого ему нужно:

1) Запустить текстовый редактор.
2) Создать новый документ (*если редактор не делает это самостоятельно*).
3) Набрать в документе текст.
4) Отформатировать текст должным образом.
5) Отправить документ на печать.
6) Сохранить документ (*спорно, но допустим*).
7) Закрыть текстовый редактор.

Вот мы и получили пользовательский сценарий, пункты которого

[76] A scenario is a hypothetical story, used to help a person think through a complex problem or system. Kaner C. An Introduction to Scenario Testing. URL: http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf.

могут стать основой для шагов тест-кейса или целого набора отдельных тест-кейсов.

Сценарии могут быть достаточно длинными и сложными, могут содержать внутри себя циклы и условные ветвления, но при всём этом они обладают рядом весьма интересных преимуществ:

- Сценарии показывают реальные и понятные примеры использования продукта (в отличие от обширных чек-листов, где смысл отдельных пунктов может теряться).
- Сценарии понятны конечным пользователям и хорошо подходят для обсуждения и совместного улучшения.
- Сценарии и их части легче оценивать с точки зрения важности, чем отдельные пункты (особенно низкоуровневых) требований.
- Сценарии отлично показывают недоработки в требованиях (если становится непонятно, что делать в том или ином пункте сценария, – с требованиями явно что-то не то).
- В предельном случае (нехватка времени и прочие форс-мажоры) сценарии можно даже не прописывать подробно, а просто именовать – и само наименование уже подскажет опытному специалисту, что делать.

Последний пункт проиллюстрируем на примере. Классифицируем потенциальных пользователей нашего приложения (напомним, что в нашем случае «пользователь» – это администратор, настраивающий работу приложения) по степени квалификации и склонности к экспериментам, а затем дадим каждому «виду пользователя» запоминающееся имя (таблица 2.t).

Таблица 2.t – Классификация пользователей

Вид пользователя	**Низкая квалификация**	**Высокая квалификация**
Не склонен к экспериментам	«Осторожный»	«Консервативный»
Склонен к экспериментам	«Отчаянный»	«Изощрённый»

Согласитесь, уже на этой стадии не составляет труда представить себе отличия в логике работы с приложением, например, «консервативного» и «отчаянного» пользователей.

Но мы пойдём дальше и озаглавим для них сами сценарии, например, в ситуациях, когда такой пользователь позитивно и негативно относится к идее внедрения нашего приложения (таблица 2.u).

Таблица 2.u – Сценарии поведения на основе классификации пользователей

Отношение пользователя	**«Осторожный»**	**«Консервативный»**	**«Отчаянный»**	**«Изощрённый»**
Позитивное	«А можно вот так?»	«Начнём с инструкции!»	«Гляньте, что я придумал!»	«Я всё оптимизирую!»
Негативное	«Я ничего не понимаю.»	«У вас вот здесь несоответствие…»	«Всё равно поломаю!»	«А я говорил!»

Проявив даже немного воображения, можно представить, что и как будет происходить в каждой из получившихся восьми ситуаций. Причём на создание пары таких таблиц уходит всего несколько минут, а эффект от их использования значительно превосходит бездумное «кликанье по кнопкам в надежде найти баг».

Куда более полное и техничное объяснение того, что такое сценарное тестирование, как его применять и выполнять должным образом, можно прочесть в статье Сэма Канера «An Introduction to Scenario Testing»[77].

Подробная классификация наборов тест-кейсов может быть выражена таблицей 2.v.

Таблица 2.v – Подробная классификация наборов тест-кейсов

По образованию тест-кейсами строгой последовательности	**По изолированности тест-кейсов друг от друга**	
	Изолированные	**Обобщённые**
Свободные	Изолированные свободные	Обобщённые свободные
Последовательные	Изолированные последовательные	Обобщённые последовательные

Следует отметить следующие особенности:

- Набор изолированных свободных тест-кейсов (рисунок 2.b): действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы можно выполнять в любом порядке.

[77] Kaner C. An Introduction to Scenario Testing. URL: http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf.

- Набор обобщённых свободных тест-кейсов (рисунок 2.c): действия из раздела «приготовления» нужно выполнить один раз (а потом просто выполнять тест-кейсы), а сами тест-кейсы можно выполнять в любом порядке.
- Набор изолированных последовательных тест-кейсов (рисунок 2.d): действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы нужно выполнять в строго определённом порядке.
- Набор обобщённых последовательных тест-кейсов (рисунок 2.e): действия из раздела «приготовления» нужно выполнить один раз (а потом просто выполнять тест-кейсы), а сами тест-кейсы нужно выполнять в строго определённом порядке.

Главное преимущество изолированности: каждый тест-кейс выполняется в «чистой среде», на него не влияют результаты работы предыдущих тест-кейсов.

Главное преимущество обобщённости: приготовления не нужно повторять (экономия времени).

Главное преимущество последовательности: ощутимое сокращение шагов в каждом тест-кейсе, т. к. результат выполнения предыдущего тест-кейса является начальной ситуацией для следующего.

Главное преимущество свободы: возможность выполнять тест-кейсы в любом порядке, а также то, что при провале некоего тест-кейса (приложение не пришло в ожидаемое состояние) остальные тест-кейсы по-прежнему можно выполнять.

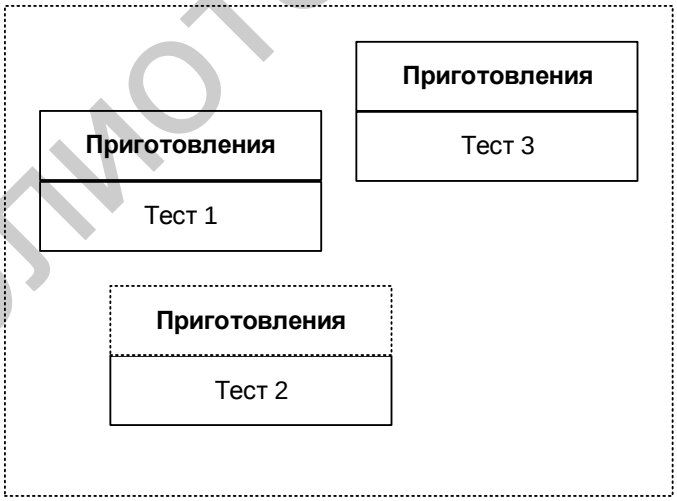


Рисунок 2.b – Набор изолированных свободных тест-кейсов

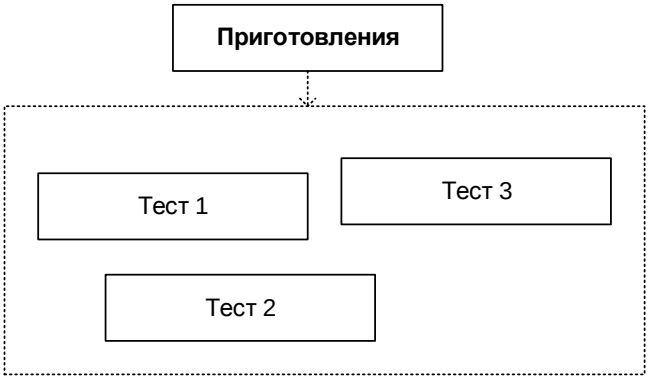


Рисунок 2.c – Набор обобщённых свободных тест-кейсов

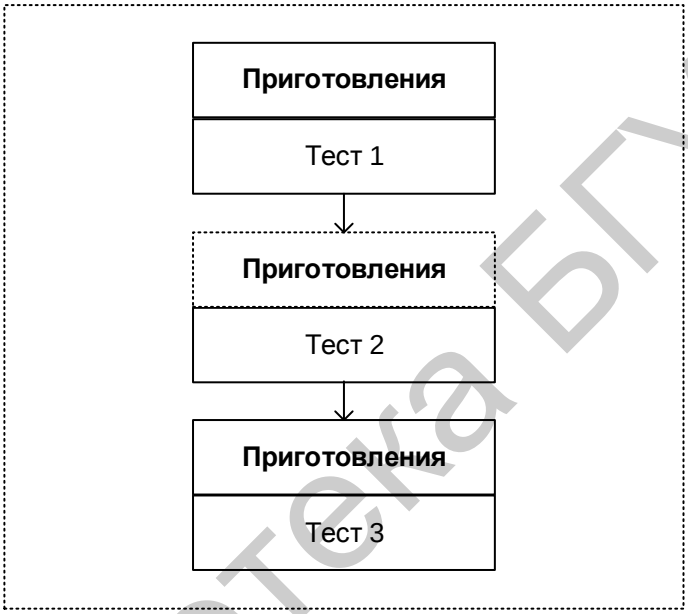


Рисунок 2.d – Набор изолированных последовательных тест-кейсов

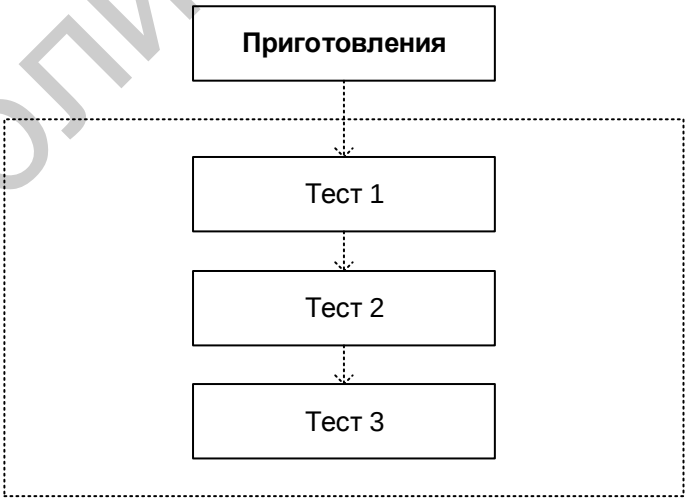


Рисунок 2.e – Набор обобщённых последовательных тест-кейсов

Ниже будут рассмотрены принципы построения наборов тест-кейсов.

Главный вопрос: как формировать наборы тест-кейсов? Правильный ответ звучит очень кратко: логично. И это не шутка. Единственная задача наборов – повысить эффективность тестирования за счёт ускорения и упрощения выполнения тест-кейсов, увеличения глубины исследования некоей области приложения или функциональности, следования типичным пользовательским сценариям (см. «Пользовательские сценарии (сценарии использования)» в подразделе 2.5) или удобной последовательности выполнения тест-кейсов и т. д.

Набор тест-кейсов всегда создаётся с какой-то целью, на основе какой-то логики, и по этим же принципам в набор включаются тесты, обладающие подходящими свойствами.

Если же говорить о наиболее типичных подходах к составлению наборов тест-кейсов, то можно обозначить следующие:

- На основе чек-листов. Посмотрите внимательно на примеры чек-листов, которые мы разработали в соответствующем подразделе 2.1: каждый пункт чек-листа может превратиться в несколько тест-кейсов – и вот мы получаем готовый набор.
- На основе разбиения приложения на модули и подмодули (см. «Модуль и подмодуль приложения» в подразделе 2.3). Для каждого модуля (или его отдельных подмодулей) можно составить свой набор тест-кейсов.
- По принципу проверки самых важных, менее важных и всех остальных функций приложения (именно по этому принципу мы составляли примеры чек-листов в вышеупомянутом подразделе 2.1).
- По принципу группировки тест-кейсов для проверки некоего уровня требований или типа требований (см. подраздел 1.4), группы требований или отдельного требования.
- По принципу частоты обнаружения тест-кейсами дефектов в приложении (например, мы видим, что некоторые тест-кейсы раз за разом завершаются неудачей, значит, мы можем объединить их в набор, условно названный «проблемные места в приложении»).
- По архитектурному принципу: наборы для проверки пользовательского интерфейса и всего уровня представления, для проверки уровня бизнес-логики, для проверки уровня данных.
- По области внутренней работы приложения, например: «тест-кейсы, затрагивающие работу с базой данных», «тест-кейсы, затрагивающие работу с файловой системой», «тест-кейсы, затрагивающие работу с сетью» и т. д.
- По видам тестирования.

Не нужно заучивать этот список. Это всего лишь примеры – грубо говоря, «первое, что приходит в голову». Важен принцип: если вы види-

те, что выполнение некоторых тест-кейсов в виде набора принесёт вам пользу, создавайте такой набор.

Примечание – Без хороших инструментальных средств управления тест-кейсами работать с наборами тест-кейсов крайне тяжело, т. к. приходится самостоятельно следить за приготовлениями, «недостающими шагами», изолированностью или обобщённостью, свободностью или последовательностью и т. д.

2.6 Контрольные вопросы и задания

- Какие разновидности тестов вы запомнили?
- Какие рекомендации по разработке тестов вы запомнили?
- Назовите основные шаги разработки тестов.
- Назовите несколько техник ускорения написания тестов.
- Каковы основные задачи планирования тестовых испытаний и составления тестового плана?
- Что такое тестовый план?
- Каковы основные свойства тестового плана?
- Что такое тестовый сценарий (test case)?
- Зачем нужны тест-кейсы?
- Каковы критерии хорошего тестового сценария?
- Каковы основные задачи планирования тестовых испытаний и составления тестового плана?
- Какие основные действия необходимо выполнить на стадии планирования?
- Перечислите артефакты, создаваемые на стадии планирования тестирования.
- Каковы основные сложности планирования тестовых испытаний?
- Назовите основные разделы тестового плана. Что описывается в каждом из них?
- Что приводится в разделе «описание процесса тестирования» тест-плана?
- Какая информация содержится в разделе «краткое описание» тест-плана?
- Какие риски необходимо учитывать при планировании тестовых испытаний?
- Каковы критерии хорошего тестового плана?
- Каковы преимущества хорошего тестового плана?
- Что такое позитивные и негативные тесты и зачем они нужны?

3 ОТЧЁТЫ О ДЕФЕКТАХ

3.1 Ошибки, дефекты, сбои, отказы

Упрощённый взгляд на понятие дефекта

В этой главе мы будем изучать терминологию (она действительно важна!), а потому начнём с очень простого: дефектом упрощённо можно считать любое расхождение ожидаемого (свойства, результата, поведения и т. д., которое мы ожидали увидеть) и фактического (свойства, результата, поведения и т. д., которое мы на самом деле увидели). При обнаружении дефекта создаётся отчёт о дефекте.

Дефект – расхождение ожидаемого и фактического результата.
Ожидаемый результат – поведение системы, описанное в требованиях.
Фактический результат – поведение системы, наблюдаемое в процессе тестирования.

ВАЖНО! Эти три определения приведены в предельно упрощённой (и даже искажённой) форме с целью первичного ознакомления. Полноценные формулировки см. далее в данной главе.

Поскольку столь простая трактовка не покрывает все возможные формы проявления проблем с программными продуктами, мы сразу же переходим к более подробному рассмотрению соответствующей терминологии.

Попробуем широко взглянуть на терминологию, описывающую проблемы. Разберёмся с широким спектром синонимов, которыми обозначают проблемы с программными продуктами и иными артефактами и процессами, сопутствующими их разработке.

В силлабусе ISTQB написано[78], что человек совершает ошибки, которые приводят к возникновению дефектов в коде, которые, в свою очередь, приводят к сбоям и отказам приложения (однако сбои и отказы могут возникать и из-за внешних условий, таких как электромагнитное воздействие на оборудование и т. д.)

[78] A human being can make an error (mistake), which produces a defect (fault, bug) in the program code, or in a document. If a defect in code is executed, the system may fail to do what it should do (or do something it shouldn't), causing a failure. Defects in software, systems or documents may result in failures, but not all defects do so. Defects occur because human beings are fallible and because there is time pressure, complex code, complexity of infrastructure, changing technologies, and/or many system interactions. Failures can be caused by environmental conditions as well. For example, radiation, magnetism, electronic fields, and pollution can cause faults in firmware or influence the execution of software by changing the hardware conditions. ISTQB Syllabus.

Таким образом, упрощённо можно изобразить схему, показанную на рисунке 3.a.

Рисунок 3.a – Ошибки, дефекты, сбои и отказы

Если же посмотреть на англоязычную терминологию, представленную в глоссарии ISTQB и иных источниках, можно построить чуть более сложную схему (рисунок 3.b).

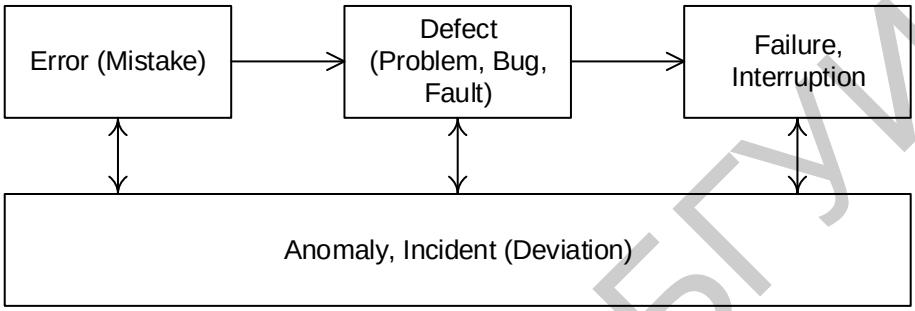


Рисунок 3.b – Взаимосвязь проблем в разработке программных продуктов

Рассмотрим все соответствующие термины.

!!! **Ошибка** (error[79], mistake) – действие человека, приводящее к некорректным результатам.

Этот термин очень часто используют как наиболее универсальный, описывающий любые проблемы («ошибка человека», «ошибка в коде», «ошибка в документации», «ошибка выполнения операции», «ошибка передачи данных», «ошибочный результат» и т. п.) Более того, куда чаще вы сможете услышать «отчёт об ошибке», чем «отчёт о дефекте». И это нормально, так сложилось исторически, к тому же термин «ошибка» на самом деле имеет очень широкий смысл.

!!! **Дефект** (defect[80], bug, problem, fault) – недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.

[79] **Error, Mistake.** A human action that produces an incorrect result. ISTQB Glossary.

[80] **Defect, Bug, Problem, Fault.** A flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system. ISTQB Glossary.

Этот термин также понимают достаточно широко, говоря о дефектах в документации, настройках, входных данных и т. д. Почему глава называется именно «отчёты о дефектах»? Потому что этот термин как раз стоит посередине – бессмысленно писать отчёты о человеческих ошибках, равно как и почти бесполезно просто описывать проявления сбоев и отказов – нужно «докопаться» до их причины, и первым шагом в этом направлении является именно описание дефекта.

!!!	**Сбой** (interruption[81]) или **отказ** (failure[82]) – отклонение поведения системы от ожидаемого. В ГОСТ 27.002-89 даны хорошие и краткие определения сбоя и отказа. **Сбой** – самоустраняющийся отказ или однократный отказ, устраняемый незначительным вмешательством оператора. **Отказ** – событие, заключающееся в нарушении работоспособного состояния объекта.

Эти термины скорее относятся к теории надёжности и нечасто встречаются в повседневной работе тестировщика, но именно сбои и отказы являются тем, что тестировщик замечает в процессе тестирования (и отталкиваясь от чего, проводит исследование с целью выявить дефект и его причины).

!!!	**Аномалия** (anomaly[83]) или **инцидент** (incident[84], deviation[84]) – любое отклонение наблюдаемого (фактического) состояния, поведения, значения, результата, свойства от ожиданий наблюдателя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

Ошибки, дефекты, сбои, отказы и т. д. являются проявлением аномалий – отклонений фактического результата от ожидаемого. Стоит отметить, что ожидаемый результат действительно может основываться на опыте и здравом смысле, т. к. поведение программного средства ни-

[81] **Interruption.** A suspension of a process, such as the execution of a computer program, caused by an event external to that process and performed in such a way that the process can be resumed. URL: http://www.electropedia.org/iev/iev.nsf/display?openform&ievref=714-22-10.

[82] **Failure.** Deviation of the component or system from its expected delivery, service or result. ISTQB Glossary.

[83] **Anomaly.** Any condition that deviates from expectation based on requirements specifications, design documents, user documents, standards, etc. or from someone's perception or experience. Anomalies may be found during, but not limited to, reviewing, testing, analysis, compilation, or use of software products or applicable documentation. See also bug, defect, deviation, error, fault, failure, incident, problem. ISTQB Glossary.

[84] **Incident, Deviation.** Any event occurring that requires investigation. ISTQB Glossary.

когда не специфицируют до уровня базовых элементарных приёмов работы с компьютером.

Теперь, чтобы окончательно избавиться от путаницы и двусмысленности, договоримся, что именно мы будем считать дефектом в контексте данной книги.

Дефект – отклонение (deviation[84]) фактического результата (actual result[85]) от ожиданий наблюдателя (expected result[86]), сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

Отсюда логически вытекает, что дефекты могут встречаться не только в коде приложения, но и в любой документации, в архитектуре и дизайне, в настройках – где угодно.

Важно понимать, что приведённое определение дефекта позволяет лишь поднять вопрос о том, является ли некое поведение приложения дефектом. В случае, если из проектной документации не следует однозначного положительного ответа, обязательно стоит обсудить свои выводы с коллегами и добиться донесения поднятого вопроса до заказчика, если его мнение по обсуждаемому «кандидату в баги» неизвестно.

Хорошее представление о едва-едва затронутой нами теме теории надёжности можно получить, прочитав книгу Рудольфа Стапелберга «Руководство по надёжности, доступности, ремонтопригодности и безопасности в инженерном проектировании» (Stapelberg R. F. Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design).

А краткую, но достаточно подробную классификацию аномалий в программных продуктах можно посмотреть в стандарте IEEE 1044:2009 Standard Classification For Software Anomalies.

3.2 Отчёт о дефекте и его жизненный цикл

Как было сказано в подразделе 3.1, при обнаружении дефекта тестировщик создаёт отчёт о дефекте.

[85] **Actual result.** The behavior produced/observed when a component or system is tested. ISTQB Glossary.

[86] **Expected result, Expected outcome, Predicted outcome.** The behavior predicted by the specification, or another source, of the component or system under specified conditions. ISTQB Glossary.

Отчёт о дефекте (defect report[87]) – документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.

Как следует из самого определения, отчёт о дефекте пишется со следующими основными целями:

- Предоставить информацию о проблеме – уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы.
- Приоритизировать проблему – определить степень опасности проблемы для проекта и желаемые сроки её устранения.
- Содействовать устранению проблемы – качественный отчёт о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

На последней цели следует остановиться подробнее. Есть мнение, что «хорошо написанный отчёт о дефекте – половина решения проблемы для программиста». И действительно, как мы увидим далее, от полноты, корректности, аккуратности, подробности и логичности отчёта о дефекте зависит очень многое – одна и та же проблема может быть описана так, что программисту останется буквально исправить пару строк кода, а может быть описана и так, что сам автор отчёта на следующий день не сможет понять, что же он имел в виду.

ВАЖНО! «Сверхцель» написания отчёта о дефекте состоит в быстром исправлении ошибки (а в идеале – и недопущении её возникновения в будущем). Потому качеству отчётов о дефекте следует уделять особое, повышенное внимание.

Отчёт о дефекте (и сам дефект вместе с ним) проходит определённые стадии жизненного цикла, которые схематично можно показать так (рисунок 3.c):

- Обнаружен (submitted) – начальное состояние отчёта (иногда называется «Новый» (new)), в котором он находится сразу после создания. Некоторые средства также позволяют сначала создавать черновик (draft) и лишь потом публиковать отчёт.
- Назначен (assigned) – в это состояние отчёт переходит с момента, когда кто-то из проектной команды назначается ответственным за исправление дефекта. Назначение ответственного

[87] **Defect report, Bug report.** A document reporting on any flaw in a component or system that can cause the component or system to fail to perform its required function. ISTQB Glossary.

производится или решением лидера команды разработки, или коллегиально, или по добровольному принципу, или иным принятым в команде способом или выполняется автоматически на основе определённых правил.

- Исправлен (fixed) – в это состояние отчёт переводит ответственный за исправление дефекта член команды после выполнения соответствующих действий по исправлению.
- Проверен (verified) – в это состояние отчёт переводит тестировщик, удостоверившийся, что дефект на самом деле был устранён. Как правило, такую проверку выполняет тестировщик, изначально написавший отчёт о дефекте.

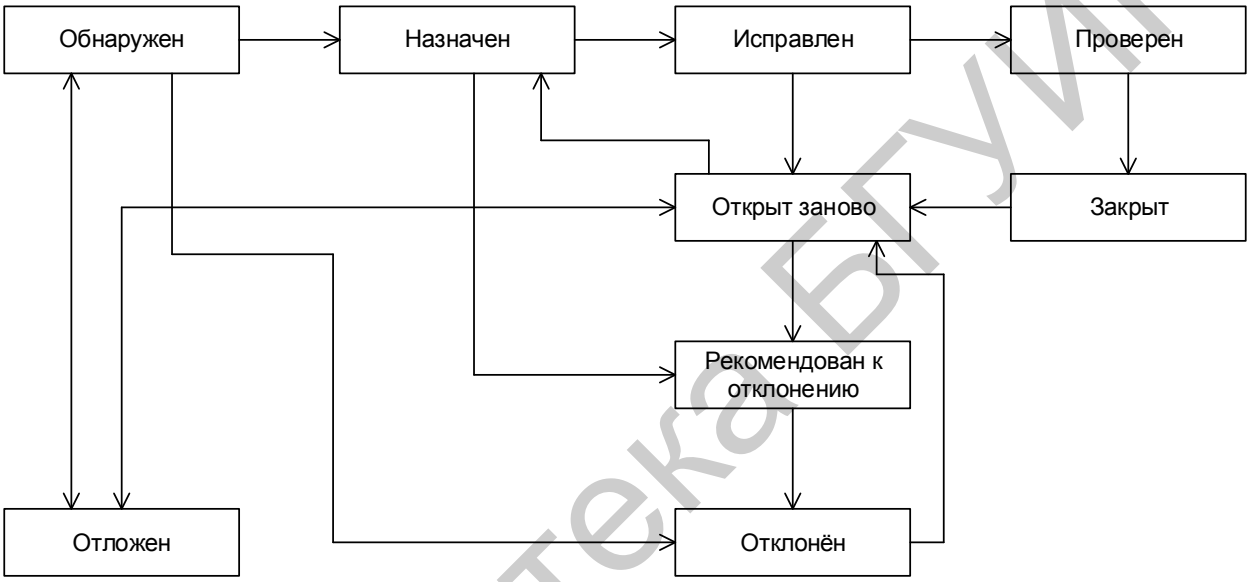


Рисунок 3.c – Жизненный цикл отчёта о дефекте с наиболее типичными переходами между состояниями

Набор стадий жизненного цикла, их наименование и принцип перехода от стадии к стадии может различаться в разных инструментальных средствах управления жизненным циклом отчётов о дефектах. Более того – многие такие средства позволяют гибко настраивать эти параметры. На рисунке 3.c показан лишь общий принцип.

- Закрыт (closed) – состояние отчёта, означающее, что по данному дефекту не планируется никаких дальнейших действий. Здесь есть некоторые расхождения в жизненном цикле, принятом в разных инструментальных средствах управления отчётами о дефектах:
  - В некоторых средствах существуют оба состояния – «Проверен» и «Закрыт», чтобы подчеркнуть, что в состоянии «Проверен» ещё могут потребоваться какие-то дополнитель-

ные действия (обсуждения, дополнительные проверки в новых билдах и т. д.), в то время как состояние «Закрыт» означает «с дефектом покончено, больше к этому вопросу не возвращаемся».

- В некоторых средствах одного из состояний нет (оно поглощается другим).
- В некоторых средствах в состояние «Закрыт» или «Отклонён» отчёт о дефекте может быть переведён из множества предшествующих состояний с резолюциями наподобие:
  - «Не является дефектом» – приложение так и должно работать, описанное поведение не является аномальным.
  - «Дубликат» – данный дефект уже описан в другом отчёте.
  - «Не удалось воспроизвести» – разработчикам не удалось воспроизвести проблему на своём оборудовании.
  - «Не будет исправлено» – дефект есть, но по каким-то серьёзным причинам его решено не исправлять.
  - «Невозможно исправить» – непреодолимая причина дефекта находится вне области полномочий команды разработчиков, например существует проблема в операционной системе или аппаратном обеспечении, влияние которой устранить разумными способами невозможно.

Как было только что подчёркнуто, в некоторых средствах отчёт о дефекте в подобных случаях будет переведён в состояние «Закрыт», в некоторых – в состояние «Отклонён», в некоторых – часть случаев закреплена за состоянием «Закрыт», часть – за «Отклонён».

- Открыт заново (reopened) – в это состояние (как правило, из состояния «Исправлен») отчёт переводит тестировщик, удостоверившийся, что дефект по-прежнему воспроизводится на билде, в котором он уже должен быть исправлен.
- Рекомендован к отклонению (to be declined) – в это состояние отчёт о дефекте может быть переведён из множества других состояний с целью вынести на рассмотрение вопрос об отклонении отчёта по той или иной причине. Если рекомендация является обоснованной, отчёт переводится в состояние «Отклонён» (см. следующий пункт).
- Отклонён (declined) – в это состояние отчёт переводится в случаях, подробно описанных в пункте «Закрыт», если средство управления отчётами о дефектах предполагает использование это-

го состояния вместо состояния «Закрыт» для тех или иных резолюций по отчёту.

- Отложен (deferred) – в это состояние отчёт переводится в случае, если исправление дефекта в ближайшее время является нерациональным или не представляется возможным, однако есть основания полагать, что в обозримом будущем ситуация исправится (выйдет новая версия библиотеки, вернётся из отпуска специалист по некоей технологии, изменятся требования заказчика и т. д.)

Для полноты рассмотрения данной подтемы приведён пример жизненного цикла, принятого по умолчанию в инструментальном средстве управления отчётами о дефектах JIRA[88] (рисунок 3.d).

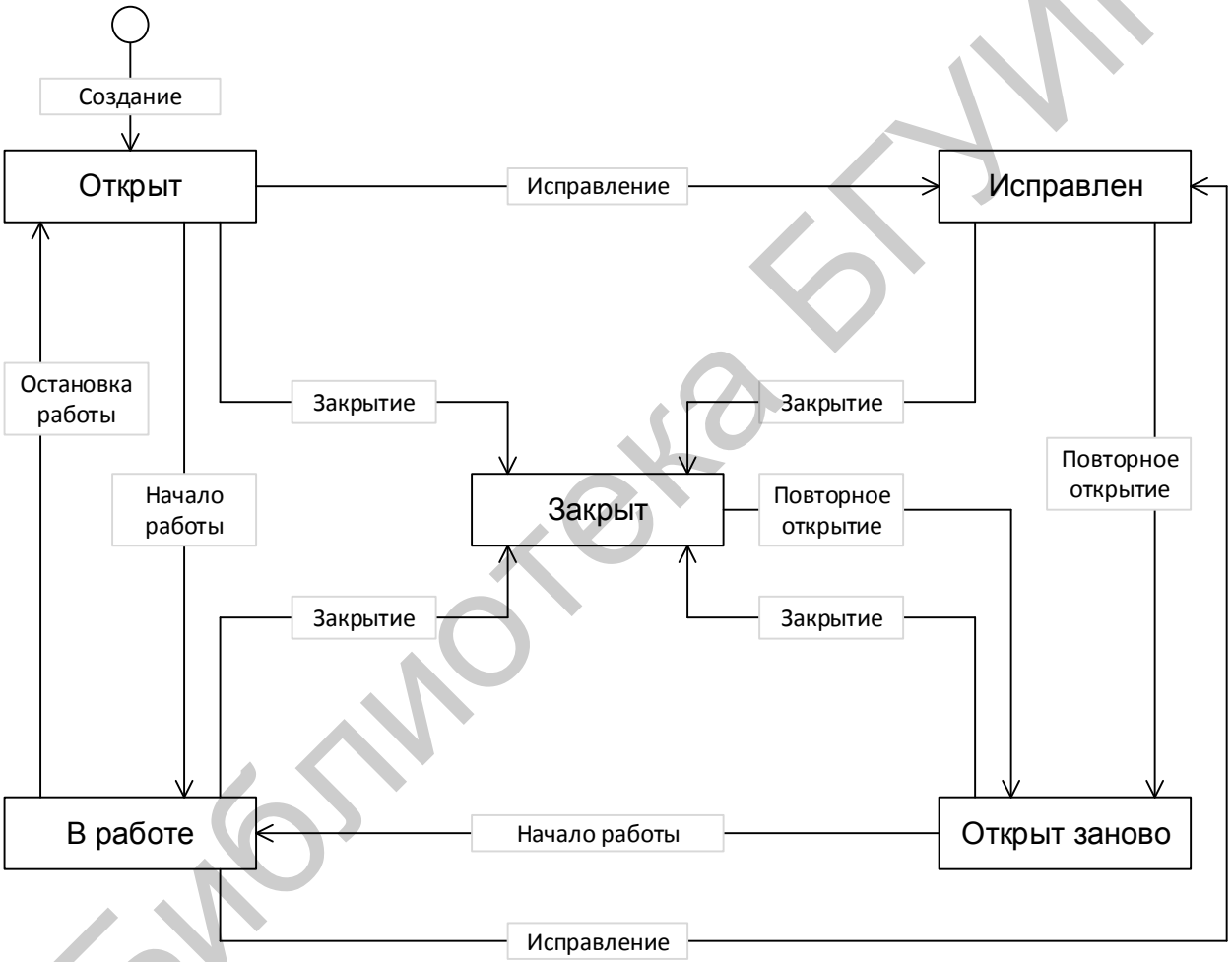


Рисунок 3.d – Жизненный цикл отчёта о дефекте в JIRA

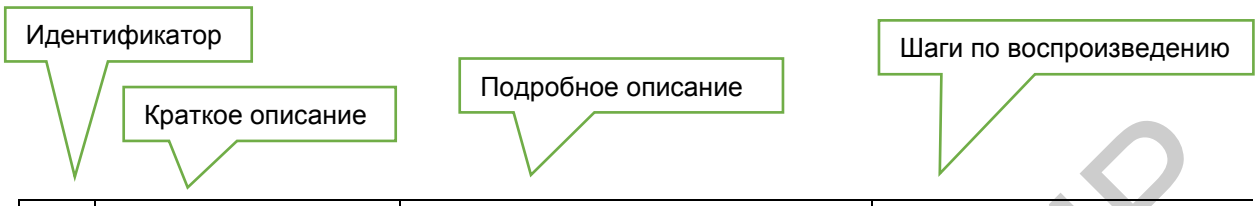
3.3 Атрибуты (поля) отчёта о дефекте

В зависимости от инструментального средства управления отчёта-

[88] What is Workflow. URL: https://confluence.atlassian.com/display/JIRA/What+is+Workflow.

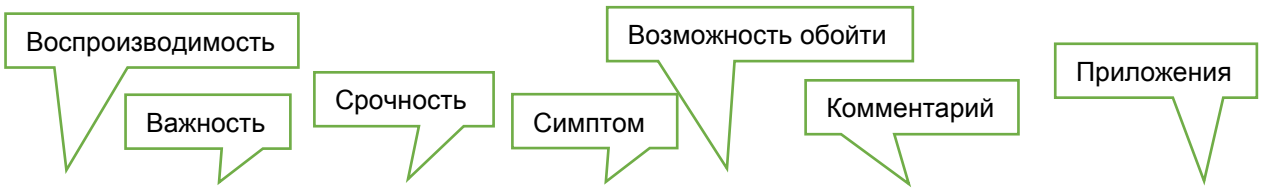
ми о дефектах внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

Общий вид всей структуры отчёта о дефекте представлен на рисунке 3.е.



19	Бесконечный цикл обработки входного файла с атрибутом «только для чтения»	Если у входного файла выставлен атрибут «только для чтения», после обработки приложению не удаётся переместить его в каталог-приёмник: создаётся копия файла, но оригинал не удаляется, и приложение снова и снова обрабатывает этот файл и безуспешно пытается переместить его в каталог-приёмник. **Ожидаемый результат:** после обработки файл перемещён из каталога-источника в каталог-приёмник. **Фактический результат:** обработанный файл копируется в каталог-приёмник, но его оригинал остаётся в каталоге-источнике. **Требование:** ДС-2.1.	1. Поместить в каталог-источник файл допустимого типа и размера. 2. Установить данному файлу атрибут «только для чтения». 3. Запустить приложение. Дефект: обработанный файл появляется в каталоге-приёмнике, но не удаляется из каталога-источника, файл в каталоге-приёмнике непрерывно обновляется (видно по значению времени последнего изменения).

Рисунок 3.е – Общий вид отчёта о дефекте
(продолжение рисунка см. на с. 76)



Всегда	Средняя	Обычная	Некорректная операция	Нет	Если заказчик не планирует использовать установку атрибута «только для чтения» файлам в каталоге-источнике для достижения неких своих целей, можно просто снимать этот атрибут и спокойно перемещать файл	-

Рисунок 3.е – Продолжение (начало см. на с. 75)

Задание 3.а: как вы думаете, почему этот отчёт о дефекте можно по формальным признакам отклонить с резолюцией «не является дефектом»?

Теперь рассмотрим каждый атрибут подробно.

Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один отчёт о дефекте от другого и используемое во всевозможных ссылках. В общем случае идентификатор отчёта о дефекте может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления отчётами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить суть дефекта и часть приложения (или требований), к которой он относится.

Краткое описание (summary) должно в предельно лаконичной форме давать исчерпывающий ответ на вопросы «Что произошло?» «Где это произошло»? «При каких условиях это произошло?». Например: «Отсутствует логотип на странице приветствия, если пользователь является администратором»:

- Что произошло? Отсутствует логотип.
- Где это произошло? На странице приветствия.
- При каких условиях это произошло? Если пользователь является администратором.

Одной из самых больших проблем для начинающих тестировщиков является именно заполнение поля «краткое описание», которое одновременно должно:

- Содержать предельно краткую, но в то же время достаточную для понимания сути проблемы информацию о дефекте.
- Отвечать на только что упомянутые вопросы («что, где и при каких условиях случилось») или как минимум на те 1–2 вопроса, которые применимы к конкретной ситуации.
- Быть достаточно коротким, чтобы полностью помещаться на экране (в тех системах управления отчётами о дефектах, где конец этого поля обрезается или приводит к появлению скроллинга).
- При необходимости содержать информацию об окружении, под которым был обнаружен дефект.
- Быть законченным предложением русского или английского (или иного) языка, построенным по соответствующим правилам грамматики.

Для создания хороших кратких описаний дефектов рекомендуется пользоваться следующим алгоритмом:

1. Полноценно понять суть проблемы. До тех пор, пока у тестировщика нет чёткого, «кристально чистого» понимания того, «что сломалось», писать отчёт о дефекте вообще едва ли стоит.
2. Сформулировать подробное описание (description) дефекта – сначала без оглядки на длину получившегося текста.
3. Убрать из получившегося подробного описания всё лишнее, уточнить важные детали.
4. Выделить в подробном описании слова (словосочетания, фрагменты фраз), отвечающие на вопросы, «что, где и при каких условиях случилось».
5. Оформить результат, получившийся в пункте 4, в виде законченного грамматически правильного предложения.
6. Если предложение получилось слишком длинным, переформулировать его, сократив длину (за счёт подбора синонимов, использования общепринятых аббревиатур и сокращений). К слову, в английском языке предложение почти всегда будет короче русского аналога.

Рассмотрим несколько примеров применения этого алгоритма.

Ситуация 1. Тестированию подвергается некое веб-приложение, поле описания товара должно допускать ввод максимум 250 символов; в процессе тестирования оказалось, что этого ограничения нет.

1. Суть проблемы: исследование показало, что ни на клиентской, ни на серверной части нет никаких механизмов, проверяющих и/или ограничивающих длину введённых в поле «О товаре» данных.

2. Исходный вариант подробного описания: в клиентской и серверной части приложения отсутствуют проверка и ограничение длины данных, вводимых в поле «О товаре» на странице http://testapplication/admin/goods/edit/.

3. Конечный вариант подробного описания:

- Фактический результат: в описании товара (поле «О товаре», http://testapplication/admin/goods/edit/) отсутствуют проверка и ограничение длины вводимого текста (максимум 250 символов).
- Ожидаемый результат: в случае попытки ввода более 251 символа выводится сообщение об ошибке.

4. Определение «что, где и при каких условиях случилось»:

- Что: отсутствуют проверка и ограничение длины вводимого текста.
- Где: описание товара, поле «О товаре», http://testapplication/admin/goods/edit/.
- При каких условиях: (в данном случае дефект присутствует всегда, вне зависимости от каких бы то ни было особых условий).

5. Первичная формулировка: отсутствуют проверка и ограничение максимальной длины текста, вводимого в поле «О товаре» описания товара.

6. Сокращение (итоговое краткое описание): нет ограничения максимальной длины поля «О товаре». Английский вариант: no check for «О товаре» max length.

Ситуация 2. Попытка открыть в приложении пустой файл приводит к краху клиентской части приложения и потере несохранённых пользовательских данных на сервере:

1. Суть проблемы: клиентская часть приложения начинает «вслепую» читать заголовок файла, не проверяя ни размер, ни корректность формата, ничего; возникает некая внутренняя ошибка, и клиентская часть приложения некорректно прекращает работу, не закрыв сессию с сервером; сервер закрывает сессию по тайм-ауту (повторный запуск клиентской части запускает новую сессию, так что старая сессия и все данные в ней в любом случае утеряны).

2. Исходный вариант подробного описания: некорректный анализ открываемого клиентом файла приводит к краху клиента и необратимой утере текущей сессии с сервером.

3. Конечный вариант подробного описания:

- Фактический результат: отсутствие проверки корректности открываемого клиентской частью приложения файла (в том числе пустого) приводит к краху клиентской части и необ-

ратимой потере текущей сессии с сервером (см. BR852345).

- Ожидаемый результат: производится анализ структуры открываемого файла; в случае обнаружения проблем отображается сообщение о невозможности открытия файла.

4. Определение «что, где и при каких условиях случилось»:
   - Что: крах клиентской части приложения.
   - Где: (конкретное место в приложении определить едва ли возможно).
   - При каких условиях: при открытии пустого или повреждённого файла.
5. Первичная формулировка: отсутствие проверки корректности открываемого файла приводит к краху клиентской части приложения и потере пользовательских данных.
6. Сокращение (итоговое краткое описание): крах клиента и потеря данных при открытии повреждённых файлов. Английский вариант: client crash and data loss on damaged/empty files opening.

Ситуация 3. Крайне редко по совершенно непонятным причинам на сайте нарушается отображение всего русского текста (как статических надписей, так и информации из базы данных, генерируемой динамически и т. д. – всё «становится вопросиками»).

1. Суть проблемы: фреймворк, на котором построен сайт, подгружает специфические шрифты с удалённого сервера; если соединение обрывается, нужные шрифты не подгружаются и используются шрифты по умолчанию, в которых нет русских символов.
2. Исходный вариант подробного описания: ошибка загрузки шрифтов с удалённого сервера приводит к использованию локальных несовместимых с требуемой кодировкой шрифтов.
3. Конечный вариант подробного описания:
   - Фактический результат: периодическая невозможность загрузить шрифты с удалённого сервера приводит к использованию локальных шрифтов, несовместимых с требуемой кодировкой.
   - Ожидаемый результат: необходимые шрифты подгружаются всегда (или используется локальный источник необходимых шрифтов).
4. Определение «что, где и при каких условиях случилось»:
   - Что: используются несовместимые с требуемой кодировкой шрифты.
   - Где: (по всему сайту).
   - При каких условиях: в случае ошибки соединения с сервером, с которого подгружаются шрифты.
5. Первичная формулировка: периодические сбои внешнего источника шрифтов приводят к сбою отображения русского текста.

6. Сокращение (итоговое краткое описание): неверный показ русского текста при недоступности внешних шрифтов. Английский вариант: wrong presentation of Russian text in case of external fonts inaccessibility.

Для закрепления материала ещё раз представим эти три ситуации в виде таблицы 3.а.

Таблица 3.а – Проблемные ситуации и формулировки кратких описаний дефектов

Ситуация	**Русский вариант краткого описания**	**Английский вариант краткого описания**
Тестированию подвергается некое веб-приложение, поле описания товара должно допускать ввод максимум 250 символов; в процессе тестирования оказалось, что этого ограничения нет	Нет ограничения максимальной длины поля «О товаре»	No check for «О товаре» max length
Попытка открыть в приложении пустой файл приводит к краху клиентской части приложения и потере несохранённых пользовательских данных на сервере	Крах клиента и потеря данных при открытии повреждённых файлов	Client crash and data loss on damaged/empty files opening
Крайне редко по совершенно непонятным причинам на сайте нарушается отображение всего русского текста (как статических надписей, так и данных из базы данных, генерируемых динамически и т. д. – всё «становится вопросиками»)	Неверный показ русского текста при недоступности внешних шрифтов	Wrong presentation of Russian text in case of external fonts inaccessibility

Возвращаемся к рассмотрению полей отчёта о дефекте.

Подробное описание (description) представляет в развёрнутом виде необходимую информацию о дефекте, а также (обязательно!) описание фактического результата, ожидаемого результата и ссылку на требование (если это возможно).

Рассмотрим пример подробного описания:

Если в систему входит администратор, на странице приветствия отсутствует логотип.
Фактический результат: логотип отсутствует в левом верхнем углу страницы.
Ожидаемый результат: логотип отображается в левом верхнем углу страницы.
Требование: R245.3.23b.

В отличие от краткого описания, которое, как правило, является одним предложением, здесь можно и нужно давать подробную информацию. Если одна и та же проблема (вызванная одним источником) проявляется в нескольких местах приложения, можно в подробном описании перечислить эти места.

Шаги по воспроизведению (steps to reproduce, STR) описывают действия, которые необходимо выполнить для воспроизведения дефекта. Это поле похоже на шаги тест-кейса, за исключением одного важного отличия: здесь действия прописываются максимально подробно, с указанием конкретных вводимых значений и самых мелких деталей, т. к. отсутствие этой информации в сложных случаях может привести к невозможности воспроизведения дефекта.

Рассмотрим пример шагов воспроизведения:

1. Открыть http://testapplication/admin/login/.
2. Авторизоваться с именем «defaultadmin» и паролем «dapassword».

Дефект: в левом верхнем углу страницы отсутствует логотип (вместо него отображается пустое пространство с надписью «logo»).

Воспроизводимость (reproducibility) показывает, при каждом ли прохождении по шагам воспроизведения дефекта удаётся вызвать его проявление. Это поле принимает всего два значения: всегда (always) или иногда (sometimes).

Можно сказать, что воспроизводимость «иногда» означает, что тестировщик не нашёл настоящую причину возникновения дефекта. Это приводит к серьёзным дополнительным сложностям в работе с дефектом:

- Тестировщику нужно потратить много времени на то, чтобы удостовериться в наличии дефекта (т. к. однократный сбой в работе приложения мог быть вызван огромным количеством посторонних причин).

- Разработчику тоже нужно потратить время, чтобы добиться проявления дефекта и убедиться в его наличии. После внесения исправлений в приложение разработчик фактически должен полагаться только на свой профессионализм, т. к. даже многократное прохождение по шагам воспроизведения в таком случае не гарантирует, что дефект был исправлен (возможно, через ещё 10–20 повторений он бы проявился).
- Тестировщику, верифицирующему исправление дефекта и вовсе остаётся верить разработчику на слово по той же самой причине: даже если он попытается воспроизвести дефект 100 раз и потом прекратит попытки, может так случиться, что на 101-й раз дефект всё же воспроизвёлся бы.

Как легко догадаться, такая ситуация является крайне неприятной, а потому рекомендуется один раз потратить время на тщательную диагностику проблемы, найти её причину и перевести дефект в разряд воспроизводимых всегда.

Важность (severity) показывает степень ущерба, который наносится проекту существованием дефекта.

В общем случае выделяют следующие градации важности:

- Критическая (critical) – существование дефекта приводит к масштабным последствиям катастрофического характера, например: потеря данных, раскрытие конфиденциальной информации, нарушение ключевой функциональности приложения и т. д.
- Высокая (major) – существование дефекта приносит ощутимые неудобства многим пользователям в рамках их типичной деятельности, например: недоступность вставки из буфера обмена, неработоспособность общепринятых клавиатурных комбинаций, необходимость перезапуска приложения при выполнении типичных сценариев работы.
- Средняя (medium) – существование дефекта слабо влияет на типичные сценарии работы пользователей, и/или существует обходной путь достижения цели, например: диалоговое окно не закрывается автоматически после нажатия кнопок «OK»/«Cancel», при распечатке нескольких документов подряд не сохраняется значение поля «Двусторонняя печать», перепутаны направления сортировок по некоему полю таблицы.
- Низкая (minor) – существование дефекта редко обнаруживается незначительным процентом пользователей и (почти) не влияет на их работу, например: опечатка в глубоко вложенном пункте меню настроек, некое окно сразу при отображении расположено неудобно (нужно перетянуть его в удобное место), неточно отображается время до завершения операции копирования файлов.

Срочность (priority) показывает, как быстро дефект должен быть устранён.

В общем случае выделяют следующие градации срочности:

- Наивысшая (ASAP, as soon as possible) срочность указывает на необходимость устранить дефект настолько быстро, насколько это возможно. В зависимости от контекста «настолько быстро, насколько возможно» может варьироваться от «в ближайшем билде» до единиц минут.
- Высокая (high) срочность означает, что дефект следует исправить вне очереди, т. к. его существование или уже объективно мешает работе, или начнёт создавать такие помехи в самом ближайшем будущем.
- Обычная (normal) срочность означает, что дефект следует исправить в порядке общей очерёдности. Такое значение срочности получает большинство дефектов.
- Низкая (low) срочность означает, что в обозримом будущем исправление данного дефекта не окажет существенного влияния на повышение качества продукта.

Несколько дополнительных рассуждений о важности и срочности стоит рассмотреть отдельно.

Один из самых частых вопросов относится к тому, какая между ними связь. Никакой. Для лучшего понимания этого факта можно сравнить важность и срочность с координатами X и Y точки на плоскости. Хоть «на бытовом уровне» и кажется, что дефект с высокой важностью следует исправить в первую очередь, в реальности ситуация может выглядеть совсем иначе.

Чтобы проиллюстрировать эту мысль подробнее, вернёмся к перечню градаций: заметили ли вы, что для разных степеней важности примеры приведены, а для разных степеней срочности – нет? И это не случайно.

Зная суть проекта и суть дефекта, его важность определить достаточно легко, т. к. мы можем проследить влияние дефекта на критерии качества, степень выполнения требований той или иной важности и т. д. Но срочность исправления дефекта можно определить только в конкретной ситуации.

Поясним на жизненном примере: насколько для жизни человека важна вода? Очень важна, без воды человек умирает. Значит, важность воды для человека можно оценить как критическую. Но можем ли мы ответить на вопрос «Как быстро человеку нужно выпить воды?», не зная, о какой ситуации идёт речь? Если рассматриваемый человек умирает от жажды в пустыне, срочность будет наивысшей. Если он просто сидит в офисе и думает, не попить ли чая, срочность будет обычной или даже низкой.

Вернёмся к примерам из разработки программного обеспечения и покажем четыре случая сочетания важности и срочности в таблице 3.b.

Таблица 3.b – Примеры сочетания важности и срочности дефектов

Срочность	**Важность**	
	Критическая	**Низкая**
Наивысшая	Проблемы с безопасностью во введённом в эксплуатацию банковском ПО	На корпоративном сайте повредилась картинка с фирменным логотипом
Низкая	В самом начале разработки проекта обнаружена ситуация, при которой могут быть повреждены или вовсе утеряны пользовательские данные	В руководстве пользователя обнаружено несколько опечаток, не влияющих на смысл текста

Симптом (symptom) – позволяет классифицировать дефекты по их типичному проявлению. Не существует никакого общепринятого списка симптомов. Более того, далеко не в каждом инструментальном средстве управления отчётами о дефектах есть такое поле, а там, где оно есть, его можно настроить. В качестве примера рассмотрим следующие значения симптомов дефекта:

- Косметический дефект (cosmetic flaw) – визуально заметный недостаток интерфейса, не влияющий на функциональность приложения (например, надпись на кнопке выполнена шрифтом не той гарнитуры).
- Повреждение/потеря данных (data corruption/loss) – в результате возникновения дефекта искажаются, уничтожаются (или не сохраняются) некоторые данные (например, при копировании файлов копии оказываются повреждёнными).
- Проблема в документации (documentation issue) – дефект относится не к приложению, а к документации (например, отсутствует раздел руководства по эксплуатации).
- Некорректная операция (incorrect operation) – некоторая операция выполняется некорректно (например, калькулятор показывает ответ 17 при умножении 2 на 3).
- Проблема инсталляции (installation problem) – дефект проявляется на стадии установки и/или конфигурирования приложения.
- Ошибка локализации (localization issue) – что-то в приложении не переведено или переведено неверно на выбранный язык интерфейса.
- Нереализованная функциональность (missing feature) –

некая функция приложения не выполняется или не может быть вызвана (например, в списке форматов для экспорта документа отсутствует несколько пунктов, которые там должны быть).

- Проблема масштабируемости (scalability) – при увеличении количества доступных приложению ресурсов не происходит ожидаемого прироста производительности приложения).
- Низкая производительность (slow performance) – выполнение неких операций занимает недопустимо большое время.
- Крах системы (system crash) – приложение прекращает работу или теряет способность выполнять свои ключевые функции (также может сопровождаться крахом операционной системы, веб-сервера и т. д.).
- Неожиданное поведение (unexpected behavior) – в процессе выполнения некоторой типичной операции приложение ведёт себя необычным (отличным от общепринятого) образом (например, после добавления в список новой записи активной становится не новая запись, а первая в списке).
- Недружественное поведение (unfriendly behavior) – поведение приложения создаёт пользователю неудобства в работе (например, на разных диалоговых окнах в разном порядке расположены кнопки «OK» и «Cancel»).
- Расхождение с требованиями (variance from specs) – этот симптом указывают, если дефект сложно соотнести с другими симптомами, но тем не менее приложение ведёт себя не так, как описано в требованиях.
- Предложение по улучшению (enhancement) – во многих инструментальных средствах управления отчётами о дефектах для этого случая есть отдельный вид отчёта, т. к. предложение по улучшению формально нельзя считать дефектом: приложение ведёт себя согласно требованиям, но у тестировщика есть обоснованное мнение о том, как ту или иную функциональность можно улучшить.

Часто встречается вопрос о том, может ли у одного дефекта быть сразу несколько симптомов. Да, может. Например, крах системы очень часто ведёт к потере или повреждению данных. Но в большинстве инструментальных средств управления отчётами о дефектах значение поля «Симптом» выбирается из списка, и потому нет возможности указать два и более симптома одного дефекта. В такой ситуации рекомендуется выбирать либо симптом, который лучше всего описывает суть ситуации, либо «наиболее опасный» симптом (например, недружественное поведение, состоящее в том, что приложение не запрашивает подтверждения перезаписи существующего файла, приводит к потере данных; здесь «потеря данных» куда уместнее, чем «недружественное поведение»).

Возможность обойти (workaround) – показывает, существует ли альтернативная последовательность действий, выполнение которой позволило бы пользователю достичь поставленной цели (например, клавиатурная комбинация Ctrl+P не работает, но распечатать документ можно, выбрав соответствующие пункты в меню). В некоторых инструментальных средствах управления отчётами о дефектах это поле может просто принимать значения «Да» и «Нет», в некоторых при выборе «Да» появляется возможность описать обходной путь. Традиционно считается, что дефектам без возможности обхода стоит повысить срочность исправления.

Комментарий (comments, additional info) – может содержать любые полезные для понимания и исправления дефекта данные. Иными словами, сюда можно писать всё то, что нельзя писать в остальные поля.

Приложения (attachments) – представляет собой не столько поле, сколько список прикреплённых к отчёту о дефекте приложений (копий экрана, вызывающих сбой файлов и т. д.)

Общие рекомендации по формированию приложений таковы:

- Если вы сомневаетесь, делать или не делать приложение, лучше сделайте.
- Обязательно прилагайте так называемые «проблемные артефакты» (например, файлы, которые приложение обрабатывает некорректно).
- Если вы прилагаете копию экрана:
  - Чаще всего вам будет нужна копия активного окна (Alt+PrintScreen), а не всего экрана (PrintScreen).
  - Обрежьте всё лишее (используйте Snipping Tool или Paint в Windows, Pinta или XPaint в Linux).
  - Отметьте на копии экрана проблемные места (обведите, нарисуйте стрелку, добавьте надпись – сделайте всё необходимое, чтобы с первого взгляда проблема была заметна и понятна).
  - В некоторых случаях стоит сделать одно большое изображение из нескольких копий экрана (разместив их последовательно), чтобы показать процесс воспроизведения дефекта. Альтернативой этого решения является создание нескольких копий экрана, названных так, чтобы имена образовывали последовательность, например: br_9_sc_**01**.png, br_9_sc_**02**.png, br_9_sc_**03**.png.
  - Сохраните копию экрана в формате JPG (если важна экономия объёма данных) или PNG (если важна точная передача картинки без искажений).
- Если вы прилагаете видеоролик с записью происходящего на экране, обязательно оставляйте только тот фрагмент, который относится к описываемому дефекту (это будет буквально несколько

секунд или минут из возможных многих часов записи). Старайтесь подобрать настройки кодеков так, чтобы получить минимальный размер ролика при сохранении достаточного качества изображения.

- Поэкспериментируйте с различными инструментами создания копий экрана и записи видеороликов с происходящим на экране. Выберите наиболе удобное для вас программое обеспечение и приучите себя постоянно его использовать.

3.4 Свойства качественных отчётов о дефектах

Отчёт о дефекте может оказаться некачественным (а следовательно, вероятность исправления дефекта понизится), если в нём нарушено одно из следующих свойств.

Тщательное заполнение всех полей точной и корректной информацией. Нарушение этого свойства происходит по множеству причин: недостаточный опыт тестировщика, невнимательность, лень и т. д. Самыми яркими проявлениями такой проблемы можно считать следующие:

- Часть важных для понимания проблемы полей не заполнена. В результате отчёт превращается в набор обрывочных сведений, использовать которые для исправления дефекта невозможно.
- Предоставленной информации недостаточно для понимания сути проблемы. Например, из такого плохого подробного описания вообще не ясно, о чём идёт речь: «Приложение иногда неверно конвертирует некоторые файлы».
- Предоставленная информация является некорректной (например, указаны неверные сообщения приложения, неверные технические термины и т. д.) Чаще всего такое происходит по невнимательности (последствия ошибочного copy-paste и отсутствия финальной вычитки отчёта перед публикацией).
- «Дефект» (именно так, в кавычках) найден в функциональности, которая ещё не была объявлена как готовая к тестированию. То есть тестировщик констатирует, что неверно работает то, что и не должно было (пока!) верно работать.
- В отчёте присутствует жаргонная лексика: как в прямом смысле – нелитературные высказывания, так и некие технические жаргонизмы, понятные крайне ограниченному кругу людей. Например: «Фигово подцепились чартники». (Имелось в виду: «Не все таблицы кодировок загружены успешно».)
- Отчёт вместо описания проблемы с приложением критикует работу кого-то из участников проектной команды. Например: «Ну каким дураком надо быть, чтобы такое сделать?!»
- В отчёте упущена некая незначительная на первый взгляд, но по факту критичная для воспроизведения дефекта проблема. Чаще все-

го это проявляется в виде пропуска какого-то шага по воспроизведению, отсутствию или недостаточной подробности описания окружения, чрезмерно обобщённом указании вводимых значений и т. п.

- Отчёту выставлены неверные (как правило, заниженные) важность или срочность. Чтобы избежать этой проблемы, стоит тщательно исследовать дефект, определять его наиболее опасные последствия и аргументированно отстаивать свою точку зрения, если коллеги считают иначе.
- К отчёту не приложены необходимые копии экрана (особенно важные для косметических дефектов) или иные файлы. Классика такой ошибки: отчёт описывает неверную работу приложения с некоторым файлом, но сам файл не приложен.
- Отчёт написан безграмотно с точки зрения человеческого языка. Иногда на это можно закрыть глаза, но иногда это становится реальной проблемой, например: «Not keyboard in parameters accepting values» (это реальная цитата; и сам автор так и не смог пояснить, что же имелось в виду).

Правильный технический язык. Это свойство в равной мере относится и к требованиям, и к тест-кейсам, и к отчётам о дефектах – к любой документации, а потому не будем повторяться – см. описанное ранее в подразделе 2.1.

Сравните два подробных описания дефекта (таблица 3.c).

Таблица 3.c – Сравнение описания дефектов

Плохое подробное описание	**Хорошее подробное описание**
Когда мы как будто бы хотим убрать папку, где что-то внутри есть, оно не спрашивает, хотим ли мы	Не производится запрос подтверждения при удалении непустого подкаталога в каталоге SOURCE_DIR. **Act:** производится удаление непустого подкаталога (со всем его содержимым) в каталоге SOURCE_DIR без запроса подтверждения. **Exp:** в случае, если в каталоге SOURCE_DIR приложение обнаруживает непустой подкаталог, оно прекращает работу с выводом сообщения «Non-empty subfolder [имя подкаталога] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.c

Специфичность описания шагов. Говоря о тест-кейсах, мы подчёркивали, что в их шагах стоит придерживаться золотой середины между специфичностью и общностью. В отчётах о дефектах предпочтение, как правило, отдаётся специфичности по очень простой причине: нехватка какой-то мелкой детали может привести к невозможности воспроизведения дефекта. Потому, если у вас есть хоть малейшее сомнение в том, важна ли какая-то деталь, считайте, что она важна.

Сравните два набора шагов по воспроизведению дефекта (таблица 3.d).

Таблица 3.d – Воспроизведение описания дефектов

Недостаточно специфичные шаги	**Достаточно специфичные шаги**
1. Отправить на конвертацию файл допустимого формата и размера, в котором русский текст представлен в разных кодировках. Дефект: конвертация кодировок производится неверно	1. Отправить на конвертацию файл в формате HTML размером от 100 КБ до 1 МБ, в котором русский текст представлен в кодировках UTF8 (10 строк по 100 символов) и WIN-1251 (20 строк по 100 символов). Дефект: текст, который был представлен в UTF8, повреждён (представлен нечитаемым набором символов)

В первом случае воспроизвести дефект практически нереально, т. к. он заключается в особенностях работы внешних библиотек по определению кодировок текста в документе, в то время как во втором случае данных достаточно если и не для понимания сути происходящего (дефект на самом деле очень «хитрый»), то хотя бы для гарантированного воспроизведения и признания факта его наличия.

Ещё раз главная мысль: в отличие от тест-кейса отчёт о дефекте может обладать повышенной специфичностью, и это будет меньшей проблемой, чем невозможность воспроизведения дефекта из-за излишне обобщённого описания проблемы.

Отсутствие лишних действий и/или их длинных описаний. Чаще всего это свойство подразумевает, что не нужно в шагах по воспроизведению дефекта долго и по пунктам расписывать то, что можно заменить одной фразой (таблица 3.e).

Таблица 3.e – Лишние действия в описании дефектов

Плохо	Хорошо
1. Указать в качестве первого параметра приложения путь к папке с исходными файлами. 2. Указать в качестве второго параметра приложения путь к папке с конечными файлами. 3. Указать в качестве третьего параметра приложения путь к файлу журнала. 4. Запустить приложение. Дефект: приложение использует первый параметр командной строки и как путь к папке с исходными файлами, и как путь к папке с конечными файлами	1. Запустить приложение со всеми тремя корректными параметрами (особенно обратить внимание, чтобы SOURCE_DIR и DESTINATION_DIR не совпадали и не были вложены друг в друга в любом сочетании). Дефект: приложение прекращает работу с сообщением «SOURCE_DIR and DESTINATION_DIR may not be the same!» (судя по всему, первый параметр командной строки используется для инициализации имён обоих каталогов.)

Вторая по частоте ошибка – начало каждого отчёта о дефекте с запуска приложения и подробного описания по приведению его в то или иное состояние. Вполне допустимой практикой является написание в отчёте о дефекте приготовлений (по аналогии с тест-кейсами) или описание нужного состояния приложения в одном (первом) шаге.

Сравните ошибки в описании дефектов по таблице 3.f.

Таблица 3.f – Ошибки в описании дефектов

Плохо	Хорошо
1. Запустить приложение со всеми верными параметрами. 2. Подождать более 30 мин. 3. Передать на конвертацию файл допустимого формата и размера. Дефект: приложение не обрабатывает файл	Предусловие: приложение запущено и проработало более 30 мин. 1. Передать на конвертацию файл допустимого формата и размера. Дефект: приложение не обрабатывает файл

Отсутствие дубликатов. Когда в проектной команде работает большое количество тестировщиков, может возникнуть ситуация, при которой один и тот же дефект будет описан несколько раз разными людь-

ми. А иногда бывает так, что даже один и тот же тестировщик уже забыл, что когда-то давно уже обнаруживал некую проблему, и теперь описывает её заново. Избежать подобной ситуации позволяет следующий набор рекомендаций:

- Если вы не уверены, что дефект не был описан ранее, произведите поиск с помощью вашего инструментального средства управления жизненным циклом отчётов о дефектах.
- Пишите максимально информативные краткие описания (т. к. поиск в первую очередь проводят по ним). Если на вашем проекте накопится множество дефектов с краткими описаниями в стиле «Кнопка не работает», вы потратите очень много времени, раз за разом перебирая десятки таких отчётов в поисках нужной информации.
- Используйте по максимуму возможности вашего инструментального средства: указывайте в отчёте о дефекте компоненты приложения, ссылки на требования, расставляйте теги и т. д. – всё это позволит легко и быстро сузить в будущем круг поиска.
- Указывайте в подробном описании дефекта текст сообщений от приложения, если это возможно. По такому тексту можно найти даже тот отчёт, в котором остальная информация приведена в слишком общем виде.
- Старайтесь по возможности участвовать в так называемых «собраниях по прояснению» (clarification meetings[89]), т. к. пусть вы и не запомните каждый дефект или каждую пользовательскую историю дословно, но в нужный момент может возникнуть ощущение «что-то такое я уже слышал», которое заставит вас произвести поиск и подскажет, что именно искать.
- Если вы обнаружили какую-то дополнительную информацию, внесите её в уже существующий отчёт о дефекте (или попросите сделать это его автора), но не создавайте отдельный отчёт.

Очевидность и понятность. Описывайте дефект так, чтобы у читающего ваш отчёт не возникло ни малейшего сомнения в том, что это действительно дефект. Лучше всего это свойство достигается за счёт тщательного объяснения фактического и ожидаемого результата, а также указания ссылки на требование в поле «Подробное описание».

Сравните описание дефектов по таблице 3.g.

[89] **Clarification meeting**. A discussion that helps the customers achieve “advance clarity” – consensus on the desired behavior of each story – by asking questions and getting examples. Crispin L., Gregory J. Agile Testing.

Таблица 3.g – Очевидность и понятность в описании дефектов

Плохое подробное описание	**Хорошее подробное описание**
Приложение не сообщает об обнаруженных подкаталогах в каталоге SOURCE_DIR	Приложение не уведомляет пользователя об обнаруженных в каталоге SOURCE_DIR подкаталогах, что приводит к необоснованным ожиданиям пользователями обработки файлов в таких подкаталогах. **Act:** приложение начинает (продолжает) работу, если в каталоге SOURCE_DIR находятся подкаталоги. **Exp:** в случае если в каталоге SOURCE_DIR приложение при запуске или в процессе работы обнаруживает пустой подкаталог, оно автоматически его удаляет (**логично ли это?**), если же обнаружен непустой подкаталог, приложение прекращает работу с выводом сообщения «Non-empty subfolder [имя подкаталога] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.c

В первом случае после прочтения подробного описания очень хочется спросить: «И что? А оно разве должно уведомлять?» Второй же вариант подробного описания даёт чёткое пояснение, что такое поведение является ошибочным согласно текущему варианту требований. Более того, во втором варианте отмечен вопрос (а в идеале нужно сделать соответствующую отметку и в самом требовании), призывающий пересмотреть алгоритм корректного поведения приложения в подобной ситуации: т. е. мы не только качественно описываем текущую проблему, но и поднимаем вопрос о дальнейшем улучшении приложения.

Прослеживаемость. Из содержащейся в качественном отчёте о дефекте информации должно быть понятно, какую часть приложения, какие функции и какие требования затрагивает дефект. Лучше всего это свойство достигается правильным использованием возможностей инструментального средства управления отчётами о дефектах: указывайте

в отчёте о дефекте компоненты приложения, ссылки на требования, тест-кейсы, смежные отчёты о дефектах (похожих дефектах; зависимых и зависящих от данного дефектах), расставляйте теги и т. д.

Некоторые инструментальные средства даже позволяют строить визуальные схемы на основе таких данных, что позволяет управление прослеживаемостью даже на очень больших проектах превратить из непосильной для человека задачи в тривиальную работу.

Отдельные отчёты для каждого нового дефекта. Существует два незыблемых правила:

- В каждом отчёте описывается **ровно один** дефект (если один и тот же дефект проявляется в нескольких местах, эти проявления перечисляются в подробном описании).
- При обнаружении нового дефекта создаётся новый отчёт. **Нельзя** для описания **нового** дефекта править **старые** отчёты, переводя их в состояние «открыт заново».

Нарушение первого правила приводит к объективной путанице, которую проще всего проиллюстрировать так: представьте, что в одном отчёте описано два дефекта, один из которых был исправлен, а второй – нет. В какое состояние переводить отчёт? Неизвестно.

Нарушение второго правила вообще порождает хаос: мало того, что теряется информация о ранее возникавших дефектах, так к тому же возникают проблемы со всевозможными метриками и банальным здравым смыслом. Именно во избежание этой проблемы на многих проектах правом перевода отчёта из состояния «закрыт» в состояние «открыт заново» обладает ограниченный круг участников команды.

Соответствие принятым шаблонам оформления и традициям. Как и в случае с тест-кейсами, с шаблонами оформления отчётов о дефектах проблем не возникает: они определены имеющимся образцом или экранной формой инструментального средства управления жизненным циклом отчётов о дефектах. Но что касается традиций, которые могут различаться даже в разных командах в рамках одной компании, то единственный совет: почитайте уже готовые отчёты о дефектах, перед тем как писать свои. Это может сэкономить вам много времени и сил.

3.5 Контрольные вопросы и задания

- Дайте определение понятия «дефект».
- Что такое отчёт о дефекте?
- Назовите атрибуты отчёта о дефекте.
- Что должно быть приведено в подробном описании дефекта?
- Перечислите и опишите этапы жизненного цикла дефекта.
- Перечислите основные симптомы дефектов.

- Какие рекомендации по написанию хороших отчётов о дефектах вы знаете?
- Каковы преимущества хорошего отчёта о дефекте?
- В чём различие важности и срочности дефекта?
- Приведите несколько примеров отчётов о дефектах не из ИТ-сферы и из ИТ-сферы.
- Какова стандартная периодичность выпуска отчёта о результатах тестирования? Чем она может определяться?
- Зачем и кому нужен отчёт о результатах тестирования?
- Что такое баг-трэкинговая система и для чего она нужна?
- Каковы цели написания отчёта о результатах тестирования?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ammann, P. Introduction to Software Testing. Chapter 4. Input Space Partition Testing / P. Ammann, J. Offut. – Saarbrücken : LAP Lambert Academic Publishing, 2012. – 56 p.
2. Barker, T. T. Documentation for software and IS development. Encyclopedia of Information Systems / T. T. Barker. – Amsterdam : Elsevier Press, 2002. – P. 679–680.
3. Barker, T. T. Writing Software Documentation: A Task-Oriented Approach (Part of the Allyn & Bacon Series in Technical Communication) / T. T. Barker. – 2nd ed. – Harlow : Longman Publishing Group, 2002. – 496 p.
4. Documenting Software Architectures / P. Clements [et al.]. – Boston : Addison-Wesley Professional, 2003. – 512 p.
5. Copeland, L. A practitioner's guide to software test design / L. Copeland. – Norwood : Artech House, 2004. – 300 p.
6. Crispin, L. Agile Testing / L. Crispin, J. Gregory. – Boston : Addison-Wesley, 2009. – 533 p.
7. Hass, A. M. Guide to Advanced Software Testing / A. M. Hass. – 2nd ed. – Norwood : Artech House Publishers, 2014.– 482 p.
8. Jyoti, J. M. Software Testing and Quality Assurance / J. M. Jyoti, S. T. Bhavana. – Hoboken : John Wiley & Sons, Inc., 2005. – 441 p.
9. Kerzner, H. Project Management: A Systems Approach to Planning, Scheduling, and Controlling / H. Kerzner. – Hoboken : Wiley, 2013. – 1296 p.
10. Nageshwar, R. P. Software Testing Concepts And Tools / R. P. Nageshwar. – M. : Dreamtech, 2006. – 492 p.
11. Ауэр, К. Экстремальное программирование: постановка процесса. С первых шагов и до победного конца. Сер. Библиотека программиста / К. Ауэр, Р. Миллер. – СПб. : Питер, 2004. – 368 с.
12. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования ПО. Сер. Библиотека программиста / Б. Бейзер. – СПб. : Питер, 2004. – 320 с.
13. Бек, К. Экстремальное программирование: разработка через тестирование. Сер. Библиотека программиста / К. Бек. – СПб. : Питер, 2003. – 224 с.

14. Блэк, Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование / Р. Блэк. – М. : Лори, 2011. – 544 с.
15. Браун, К. Быстрое тестирование / К. Браун, Р. Калбертсон, Г. Кобб. – М. : Вильямс, 2002. – 384 с.
16. Вигерс, К. Разработка требований к программному обеспечению (Software Requirements, Third Ed.) / К. Вигерс, Д. Битти. – 3-е изд. – М. : Русская редакция, 2014. – 737 с.
17. Дастин, Э. Автоматизированное тестирование программного обеспечения / Э. Дастин, Д. Рэшка. – М. : Лори, 2005. – 592 с.
18. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / С. Канер, Д. Фолк, Е. К. Нгуен. – Ярославль : ДиаСофт, 2001. – 538 с.
19. Котляров, В. П. Основы тестирования программного обеспечения / В. П. Котляров. – М. : Интернет-университет информационных технологий ИНТУИТ.ру, 2006. – 285 с.
20. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. – 3-е изд. – М. : Вильямс, 2012. – 272 с.
21. Макгрегор, Д. Тестирование объектно-ориентированного программного обеспечения : практ. пособие / Д. Макгрегор, Д. Сайкс. – Ярославль : Диасофт, 2002. – 432 с.
22. Макконнелл, С. Совершенный код. Русская редакция. Сер. Мастер-класс / С. Макконнелл. – 2-е изд. – СПб. : Питер, 2005. – 896 с.
23. Роббинс, Д. Отладка приложений. Сер. Мастер / Д. Роббинс. – СПб. : BHV, 2001. – 512 с.
24. Савин, Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. – М. : Дело, 2007. – 312 с.
25. Стотлемайер, Д. Тестирование Web-приложений / Д. Стотлемайер. – М. : Кудиц-образ, 2003. – 240 с.
26. Тамре, Л. Введение в тестирование программного обеспечения / Л. Тамре. – М. : Вильямс, 2003. – 359 с.
27. Торрес, Дж. Практическое руководство по проектированию и разработке пользовательского интерфейса / Дж. Торрес. – М. : Вильямс, 2003. – 400 с.

28. Beginner’s Guide to Mobile Application Testing [Электронный ресурс]. – 2016. – Режим доступа : http://www.softwaretestinghelp.com/beginners-guide-to-mobile-application-testing/.
29. Project Lifecycle Models: How They Differ and When to Use Them [Электронный ресурс]. – 2002. – Режим доступа : http://www.business-esolutions. com/islm.htm.
30. Smoke testing and sanity testing – Quick and simple differences [Электронный ресурс]. – 2016. – Режим доступа : http://www.softwaretestinghelp.com/smoke-testing-and-sanity-testing-difference/.
31. Westfall, L. Software Requirements Engineering: What, Why, Who, When, and How / L. Westfall [Электронный ресурс]. – 2015. – Режим доступа : http://www.westfallteam.com/Papers/The_Why_What_Who_When_and_How_Of_Software_Requirements.pdf.
32. Блок-схема выбора оптимальной методологии разработки ПО [Электронный ресурс]. – 2015. – Режим доступа : http://megamozg.ru/post/23022/.
33. Blain, T. Writing Good Requirements – The Big Ten Rules / T. Blain [Электронный ресурс]. – 2006. – Режим доступа : http://tynerblain.com/blog/2006/05/25/writing-good-requirements-the-big-ten-rules/.
34. Boehm, B. A Spiral Model of Software Development and Enhancement / B. Boehm [Электронный ресурс]. – 1988. – Режим доступа : http://csse.usc.edu/csse/TECHRPTS/1988/usccse88-500/usccse88-500.pdf.
35. Boehm, B. Spiral Development: Experience, Principles, and Refinements / B. Boehm [Электронный ресурс]. – 1988. – Режим доступа : http://www.sei.cmu.edu/reports/00sr008.pdf.
36. Brandenburg, L. Requirements Gathering vs. Elicitation / L. Brandenburg [Электронный ресурс]. – 2006. – Режим доступа : http://www.bridging-the-gap.com/requirements-gathering-vs-elicitation/.
37. Cohen, J. Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.) / J. Cohen [Электронный ресурс]. – 2013. – Режим доступа : http://smartbear.com/SmartBear/media/pdfs/best-kept-secrets-of-peer-code-review.pdf.
38. Cross-Browser Testing – Overview [Электронный ресурс]. – 2016. – Режим доступа : http://support.smartbear.com /viewarticle/55299/.

39. Firesmith, D. Using V Models for Testing / D. Firesmith [Электронный ресурс]. – 2013. – Режим доступа : http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315.
40. Gelperin, D. The Growth of Software Testing / D. Gelperin, B. Hetzel [Электронный ресурс]. – 1988. – Режим доступа : http://www.clearspecs.com/downloads/ClearSpecs16V01_GrowthOfSoftwareTest.pdf.
41. Ghahrai, A. Spiral Model / A. Ghahrai [Электронный ресурс]. – 2002. – Режим доступа : http://www.testingexcellence.com/spiral-model/.
42. Hanks, B. Requirements in the Real World / B. Hanks [Электронный ресурс]. – 2002. – Режим доступа : https://classes.soe.ucsc.edu/cmps109/Winter02/notes/requirementsLecture.pdf.
43. Kaner, C. A Tutorial in Exploratory Testing / C. Kaner [Электронный ресурс]. – 2015. – Режим доступа : http://www.kaner.com/pdfs/QAIExploring.pdf.
44. Kaner, C. An Introduction to Scenario Testing / C. Kaner [Электронный ресурс]. – 2003. – Режим доступа : http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf.
45. Kuijt, R. Extended Use Case Test Design Pattern / R. Kuijt [Электронный ресурс]. – 2013. – Режим доступа : http://www.softwaretestingclass.com/positive-and-negative-testing-in-software-testing/.
46. Meerts, J. The History of Software Testing / J. Meerts [Электронный ресурс]. – Режим доступа : http://www.testingreferences.com/testinghistory.php.
47. Rangaiah, J. Behaviour driven testing an introduction / J. Rangaiah [Электронный ресурс]. – 2015. – Режим доступа : http://www.womentesters.com/behaviour-driven-testing-an-introduction/.
48. Rouse, M. Gray box testing (gray box) definition / M. Rouse [Электронный ресурс]. – 2013. – Режим доступа : http://searchsoftwarequality.techtarget.com/definition/gray-box.
49. SmartBear TestComplete user manual [Электронный ресурс]. – 2016. – Режим доступа : http://support.smartbear.com/viewarticle/55004/.
50. What are Alpha, Beta and Gamma Testing? [Электронный ресурс]. – 2012. – Режим доступа : http://www.360logica.com/blog/2012/06/what-are-alpha-beta-and-gamma-testing.htm.
51. Whittaker, J. The 7th Plague and Beyond / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/09/7th-plague-and-beyond.html.

52. Whittaker, J. The Plague of Amnesia / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/07/plague-of-amnesia.html.
53. Whittaker, J. The Plague of Blindness / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/07/plague-of-blindness.html.
54. Whittaker, J. The Plague of Boredom / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/07/plague-of-boredom.html.
55. Whittaker, J. The Plague of Entropy / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/09/plague-of-entropy.html.
56. Whittaker, J. The Plague of Homelessness / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/07/plague-of-homelessness.html.
57. Whittaker, J. The plague of repetitiveness / J. Whittaker [Электронный ресурс]. – 2009. – Режим доступа : http://googletesting.blogspot.com/2009/06/by-james.html.
58. Kelly, M. The ROI of Test Automation / M. Kelly [Электронный ресурс]. – 1999. – Режим доступа : http://www.sqetraining.com/sites/default/files/articles/XDD8502filelistfilename1_0.pdf.
59. The Return of Investment (ROI) of Test Automation / S. Münch [et al.] [Электронный ресурс]. – 2012. – Режим доступа : https://www.ispe.org/pe-ja/roi-of-test-automation.pdf.
60. Rooney, J. Root Cause Analysis for Beginners / J. Rooney, L. V. Heuvel [Электронный ресурс]. – 2012. – Режим доступа : https://www.env.nm.gov/aqb/Proposed_Regs/Part_7_Excess_Emissions/NMED_Exhibit_18-Root_Cause_Analysis_for_Beginners.pdf.
61. Garrett, T. Implementing Automated Software Testing – Continuously Track Progress and Adjust Accordingly / T. Garrett [Электронный ресурс]. – 2009. – Режим доступа : http://www.methodsandtools.com/archive/archive.php?id=94.
62. Patel, N. Test Case Point Analysis / N. Patel [Электронный ресурс]. – 2001. – Режим доступа : http://www.stickyminds.com/sites/default/files/article/file/2013/XUS373692file1_0.pdf.
63. Sherwood, G. On the Construction of Orthogonal Arrays and Covering Arrays Using Permutation Groups / G. Sherwood [Электронный ресурс]. – 2002. – Режим доступа : http://testcover.com/pub/background/cover.htm.
64. Bolton, M. Pairwise Testing / M. Bolton [Электронный ресурс]. – 2002. – Режим доступа : http://www.developsense.com/pairwiseTesting.html.

65. Kuo-Chung, T. A Test Generation Strategy for Pairwise Testing / T. Kuo-Chung, L. Yu [Электронный ресурс]. – 2002. – Режим доступа : http://www.cs.umd.edu/class/spring2003/cmsc838p/VandV/pairwise.pdf.
66. An Improved Test Generation Algorithm for Pair-Wise Testing / S. Maity [et al.] [Электронный ресурс]. – 2003. – Режим доступа : http://www.iiserpune.ac.in/~soumen/115-FA-2003.pdf.
67. Nageswaran, S. Test Effort Estimation Using Use Case Points / S. Nageswaran [Электронный ресурс]. – 2001. – Режим доступа : http://www.bfpug.com.br/Artigos/UCP/Nageswaran-Test_Effort_Estimation_Using_UCP.pdf.
68. Software Estimation Techniques – Common Test Estimation Techniques used in SDLC [Электронный ресурс]. – 2013. – Режим доступа : http://www.softwaretestingclass.com/software-estimation-techniques/.
69. Important Software Test Metrics and Measurements – Explained with Examples and Graphs [Электронный ресурс]. – 2016. – Режим доступа : [http://www.softwaretestinghelp.com/software-test-metrics-and-measurements/.

Св. план 2017, поз. 25

Учебное издание

Куликов Святослав Святославович
Данилова Галина Владимировна

ТЕСТИРОВАНИЕ ВЕБ-ОРИЕНТИРОВАННЫХ ПРИЛОЖЕНИЙ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. И. Герман*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 12.10.2017. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Ариал».
Отпечатано на ризографе. Усл. печ. л. 6,05. Уч.-изд. л. 5,0. Тираж 100 экз. Заказ 82.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6