

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

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

ЮЗАБИЛИТИ-ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

*Рекомендовано УМО по образованию
в области информатики и радиоэлектроники
для специальности 1-58 01 01 «Инженерно-психологическое обеспечение
информационных технологий», направлений специальности
1-40 05 01-09 «Информационные системы и технологии
(в обеспечении промышленной безопасности)» и
1-40 05 01-10 «Информационные системы и технологии
(в бизнес-менеджменте)» в качестве пособия*



Минск БГУИР 2017

УДК 004.415.53(076)

ББК 32.972.1я73

Ю20

Авторы:

М.М. Меженная, Т.В. Гордейчук, М.М. Борисик, О.С. Медведев, И.Ф. Киринович

Рецензенты:

кафедра интеллектуальных систем
Белорусского национального технического университета
(протокол №11 от 17.05.2016);

заведующий лабораторией биоинформатики
государственного научного учреждения «Объединенный институт
проблем информатики Национальной академии наук Беларуси»,
кандидат технических наук, доцент И. Э. Том

Юзабилити-тестирование программного обеспечения: пособие /
Ю20 М. М. Меженная [и др.] – Минск : БГУИР, 2017. – 72 с. : ил.
ISBN 978-985-543-349-2.

Пособие посвящено вопросам анализа, планирования, проведения тестовых испытаний и оценки качества программного обеспечения на всех стадиях его жизненного цикла.

УДК 004.415.53(076)

ББК 32.972.1я73

ISBN 978-985-543-349-2

© УО «Белорусский государственный
университет информатики
и радиоэлектроники», 2017

Содержание

Введение.....	4
Лабораторная работа №1	
Виды тестирования. Планирование тестирования	5
Лабораторная работа №2	
Разработка требований	15
Лабораторная работа №3	
Тестирование требований	22
Лабораторная работа №4	
Тестирование программного обеспечения: разработка тестов	30
Лабораторная работа №5	
Поиск и документирование дефектов.....	47
Лабораторная работа №6	
Документирование результатов тестирования	56
Лабораторная работа №7	
Тестирование юзабилити	62
Литература	71

Введение

Настоящее пособие направлено на получение студентами теоретических знаний и практических навыков по современным технологиям тестирования программного обеспечения и включает основные разделы знаний, необходимые для специалистов по обеспечению качества программных продуктов:

- классификация видов тестирования;
- планирование тестирования;
- разработка и тестирование требований к программному обеспечению;
- разработка тестов;
- поиск и описание дефектов;
- оценка качества и документирование результатов тестирования.

Подробно рассмотрены особенности юзабилити тестирования, направленного на повышение эффективности человеко-компьютерного взаимодействия посредством разработки эргономичных интерфейсов.

Пособие содержит базовые проверки графического интерфейса пользователя и функциональности программного обеспечения, примеры рабочей и отчетной тестовой документации, поясняющие иллюстрации.

Авторы выражают особую благодарность компании A1QA ЗАО «Технологии качества» и лично Надежде Кныш, компании «ЕРАМ Systems» и лично Святославу Куликову за обучающие материалы и консультативную помощь.

Лабораторная работа №1

Виды тестирования. Планирование тестирования

Цель: изучить классификацию видов тестирования, разработать проверки для различных видов тестирования, научиться планировать тестовые активности в зависимости от особенностей поставляемой на тестирование функциональности.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Тестирование (Testing) – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [1].

Конечной целью тестирования является предоставление пользователю качественного программного обеспечения (ПО) [2].

Качество (Quality) – степень, с которой компонент, система или процесс соответствует зафиксированным требованиям и/или ожиданиям и нуждам пользователя или заказчика [3].

Дефект (defect, bug, ошибка) – ключевой термин тестирования, означающий отклонение фактического результата от ожидаемого. Для обнаружения дефекта необходимо выполнить три условия: знать фактический результат, знать ожидаемый результат, зафиксировать факт разницы между фактическим и ожидаемым результатом.

Процесс тестирования как процесс поиска дефектов сводится к следующей последовательности действий:

1. Узнаем ожидаемый результат.
2. Узнаем фактический результат.
3. Сравниваем ожидаемый и фактический результаты.

Источником ожидаемого результата является спецификация – детальное описание того, как должно работать ПО.

В общем случае любой дефект представляет собой отклонение от спецификации. Важно обнаружить эти дефекты до того, как их найдут конечные пользователи.

Тестирование можно классифицировать по очень большому количеству признаков. Далее приведен обобщенный список видов тестирования по различным основаниям.

1. Виды тестирования в зависимости от объекта тестирования: функциональные, пограничные, нефункциональные (рисунок 1).



Рисунок 1 – Классификация видов тестирования в зависимости от объекта

Рассмотрим функциональные виды тестирования.

Функциональное тестирование (Functional Testing) – тестирование, основанное на сравнительном анализе спецификации и функциональности компонента или системы.

Тестирование безопасности (Safety Testing) – тестирование программного продукта с целью определить его способность при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью, бизнесу, программам, собственности или окружающей среде.

Тестирование защищенности (Security Testing) – тестирование с целью оценить защищенность программного продукта от внешних воздействий (от проникновений). На практике зачастую под термином тестирование безопасности понимают в том числе и тестирование защищенности.

Рассмотрим пограничные виды тестирования.

Тестирование совместимости (Compatibility Testing) – проверка работоспособности приложения в различных средах (браузеры и их версии, операционные системы, их типы, версии и разрядность). Виды тестирования совместимости: кроссбраузерное тестирование (различные браузеры или версии браузеров), кроссплатформенное тестирование (различные операционные системы или версии операционных систем).

Рассмотрим нефункциональные виды тестирования, направленные на проверку характеристик или свойств программы (внешний вид, удобство использования, скорость работы и т. п.).

Тестирование требований (Requirements Testing) – проверка требований на соответствие основным атрибутам качества.

Тестирование прототипа (Prototype Testing) – метод выявления структурных, логических ошибок и ошибок проектирования на ранней стадии развития продукта до начала фактической разработки.

Тестирование пользовательского интерфейса (GUI Testing) – тестирование, выполняемое путем взаимодействия с системой через графический интерфейс пользователя (правописание выводимой информации; расположение и выравнивание элементов GUI; соответствие названий форм/элементов GUI их назначению; унификация стиля, цвета, шрифта; окна сообщений; изменение размеров окна, поведение курсора и горячие клавиши).

Тестирование удобства использования (Usability Testing) – тестирование с целью определения степени понятности, легкости в изучении и использовании, привлекательности программного продукта для пользователя при условии использования в заданных условиях эксплуатации (на этом уровне обращают внимание на визуальное оформление, навигацию, логичность, наличие обратной связи и др.).

Тестирование доступности (Accessibility Testing) – тестирование, которое определяет степень легкости, с которой пользователи с ограниченными способностями могут использовать систему или ее компоненты.

Тестирование интернационализации (Internationalization Testing) – тестирова-

ние адаптации продукта к языковым и культурным особенностям целого ряда регионов, в которых потенциально может использоваться продукт.

Тестирование локализации (Localization Testing) – тестирование адаптации продукта к языковым и культурным особенностям конкретного региона, отличного от того, в котором разрабатывался продукт.

Тестирование производительности (Performance Testing) – процесс тестирования с целью определения производительности программного продукта. В рамках тестирования производительности выделяют нагрузочное тестирование, объемное тестирование, тестирование стабильности и надежности, стрессовое тестирование.

Нагрузочное тестирование (Performance and Load Testing) – вид тестирования производительности, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или операций, а также определения, какую нагрузку может выдержать компонент или система.

Объемное тестирование (Volume Testing) – позволяет получить оценку производительности при увеличении объемов данных в базе данных приложения.

Тестирование стабильности и надежности (Stability/Reliability Testing) – позволяет проверять работоспособность приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.

Стрессовое тестирование (Stress Testing) – вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок, или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу.

Тестирование на отказ и восстановление (Failover and Recovery Testing) – тестирование при помощи эмуляции отказов системы или реально вызываемых отказов в управляемом окружении.

Тестирование установки (Installability Testing) и лицензирования – процесс тестирования установки программного продукта. Включает формальный тест программы установки приложения (проверка пользовательского интерфейса, навигации, удобства использования, соответствия общепринятым стандартам оформления); функциональный тест программы установки; тестирование механизма лицензирования и функций защиты от пиратства; проверку стабильности приложения после установки.

2. Виды тестирования в зависимости от знания кода: белый ящик, серый ящик, черный ящик.

Белый ящик (White Box Testing) – тестирование, основанное на анализе внутренней структуры компонентов или системы (у тестировщика есть доступ к внутренней структуре и коду приложения).

Черный ящик (Black Box Testing) – тестирование системы без знания внутренней структуры и компонентов системы (у тестировщика нет доступа к внутренней структуре и коду приложения либо в процессе тестирования он не обращается к ним).

Серый ящик (Grey Box Testing) – комбинация методов белого и черного ящика, состоящая в том, что у тестировщика есть доступ только к некоторой части внутренней структуры и кода приложения.

3. Виды тестирования в зависимости от степени автоматизации: ручное, автоматизированное тестирование.

Ручное тестирование – такое тестирование, в котором тест-кейсы выполняются тестировщиком вручную без использования средств автоматизации.

Автоматизированное тестирование (Automated Testing) – набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство.

4. Виды тестирования в зависимости от степени изолированности тестируемых компонентов: модульное, интеграционное, системное тестирование.

Модульное тестирование (Unit/Component Testing) – тестируются отдельные части (модули) системы.

Интеграционное тестирование (Integration Testing) – тестируется взаимодействие между отдельными модулями.

Системное тестирование (System Testing) – тестируется работоспособность системы в целом.

5. Виды тестирования в зависимости от подготовленности: интуитивное тестирование, исследовательское тестирование, тестирование по документации.

Интуитивное тестирование выполняется без подготовки к тестам, без определения ожидаемых результатов, проектирования тестовых сценариев.

Исследовательское тестирование – метод проектирования тестовых сценариев во время выполнения этих сценариев.

Тестирование по документации – тестирование по подготовленным тестовым сценариям, руководству по осуществлению тестов.

6. Виды тестирования в зависимости от места и времени проведения тестирования: приемочное тестирование, альфа-тестирование, бета-тестирование.

Приемочное тестирование (User Acceptance Testing, UAT) – формальное тестирование по отношению к потребностям, требованиям и бизнес-процессам пользователя, проводимое с целью определения соответствия системы критериям

приемки и представления возможности пользователям, заказчикам или иным авторизованным лицам решать, принимать систему или нет.

Альфа-тестирование (Alpha Testing) – моделируемое или действительное функциональное тестирование, выполняется в организации, разрабатывающей продукт, но не проектной командой (это может быть независимая команда тестировщиков, потенциальные пользователи, заказчики). Альфа-тестирование часто применяется к коробочному программному обеспечению в качестве внутреннего приемочного тестирования.

Бета-тестирование (Beta Testing) – эксплуатационное тестирование потенциальными или существующими клиентами/заказчиками на внешней стороне (в среде, где продукт будет использоваться), никак не связанными с разработчиками, с целью определения действительно ли компонент или система удовлетворяет требованиям клиента/заказчика и вписывается в бизнес-процессы. Бета-тестирование часто проводится как форма внешнего приемочного тестирования готового программного обеспечения для того, чтобы получить отзывы рынка.

7. Виды тестирования в зависимости от глубины тестового покрытия: Smoke, MAT, AT.

Тестовое покрытие – одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

Smoke Test – поверхностное тестирование для определения пригодности сборки для дальнейшего тестирования, должно покрывать базовые функции программного обеспечения; уровень качества: Acceptable/Unacceptable.

Minimal Acceptance Test (MAT, Positive Test) – тестирование системы или ее части только на корректных данных/сценариях; уровень качества: High/Medium/Low.

Acceptance Test (AT) – полное тестирование системы или ее части как на корректных (Positive Test), так и на некорректных данных/сценариях (Negative Test); уровень качества: High/Medium / Low. Тест на этом уровне покрывает все возможные сценарии тестирования: проверку работоспособности модулей при вводе корректных значений; проверку при вводе некорректных значений; использование форматов данных, отличных от тех, которые указаны в требованиях; проверку исключительных ситуаций, сообщений об ошибках; тестирование на различных комбинациях входных параметров; проверку всех классов эквивалентности; тестирование граничных значений интервалов; сценарии, не предусмотренные спецификацией и т. д.

8. Виды тестирования в зависимости от тестовых активностей: NFT, RT, DV.

Данная классификация тестирования иначе называется видами тестирования в зависимости от ширины тестового покрытия.

Тестирование новых функциональностей (New Feature Test, NFT) – определение качества поставленной на тестирование новой функциональности, которая ранее не тестировалась. Данный тип тестирования включает в себя: проведение полного теста (AT) непосредственно новой функциональности; тестирование новой функциональности на соответствие документации; проверку всевозможных взаимодействий ранее реализованной функциональности с новыми модулями и функциями.

Регрессионное тестирование (Regression Testing, RT) проводится с целью оценки качества ранее реализованной функциональности. Включает в себя проверку стабильности ранее реализованной функциональности после внесения изменений, например добавления новой функциональности, исправление дефектов, оптимизация кода, разворачивание приложения на новом окружении. Регрессионное тестирование выполняется на уровнях MAT, AT.

Валидация дефектов (Defect Validation, DV) – проверка результатов исправления дефектов; может включать элементы регрессионного тестирования; уровень проверки не определяется.

Процесс тестирования программного продукта включает следующие этапы:

1. Планирование тестирования.
2. Разработка рабочей тестовой документации (тестов).
3. Исполнение тестирования.

Планирование тестирования включает изучение и анализ предмета тестирования, составление тест-плана. На стадии планирования тестирования перед тестировщиком стоит задача поиска компромисса между объемом тестирования, который возможен в теории, и объемом тестирования, который возможен на практике. Результирующий тест-план представляет собой документ, описывающий и регламентирующий перечень работ по тестированию, а также соответствующие техники и подходы, стратегию, области ответственности, ресурсы, расписание и ключевые даты.

Разработка тестов, как правило, выполняется до реализации программного обеспечения, основываясь исключительно на спецификации к программному продукту.

Выполнение тестирования начинается после поставки первой версии программного продукта и представляет собой практический поиск дефектов с использованием тестовой документации, составленной ранее, а также контроль исправления обнаруженных дефектов.

Для всех программных продуктов выполняют следующие типы тестов и их композиции.

Для первой поставки программного обеспечения рекомендуется проводить Smoke + NFT_{AT} готовой функциональности: поверхностное тестирование (Smoke Test) выполняется для определения пригодности сборки для дальнейшего тестирования; полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях (Acceptance Test, AT) позволяет обнаружить дефекты и внести запись о них в багтрекинг-систему.

Для последующих поставок программного обеспечения композиции тестов могут быть следующими.

Если не была добавлена новая функциональность, то DV + RT_{МАТ}. То есть, выполняется проверка исправления дефектов программистом (Defect Validation, DV), а также проверка работоспособности остальной функциональности после исправления дефектов на позитивных сценариях (Minimal Acceptance Test, MAT).

Если была добавлена новая функциональность, то Smoke + DV + NFT_{AT} + RT_{МАТ}. В частности, выполняется поверхностное тестирование (Smoke Test), проверка исправления дефектов программистом (Defect Validation, DV), тестирование новых функциональностей (New Feature Testing, NFT), проверка старых функциональностей, т. е. регрессионное тестирование (Regression Test).

Если была добавлена новая функциональность, то возможен также вариант DV + NFT_{AT} + RT_{МАТ}, т. е. без выполнения Smoke Test.

Таким образом, для второй и последующих поставок обобщенная схема композиции тестов выглядит следующим образом:

Smoke + DV + (NFT_{AT}) + RT_{МАТ}.

В зависимости от типа и специфики приложения (web, desktop, mobile) выполняют специализированные тесты (например, кроссбраузерное или кроссплатформенное тестирование, тестирование локализации и интернационализации и др.).

Практическое задание:

1. Выбрать объект реального мира (например, карандаш, стол, чашка, клавиатура, сумка и др.) с целью последующей разработки тестовых проверок для него.
2. Разработать различные проверки в соответствии с классификацией видов тестирования для выбранного объекта реального мира. Результаты внести в таблицу 1.1.

Таблица 1.1 – Тестовые проверки для различных видов тестирования

Объект тестирования: указать		
Вид тестирования	Краткое определение вида тестирования	Тестовые проверки
Functional Testing		
Safety Testing		
Security Testing		
Compatibility Testing		
GUI Testing		
Usability Testing		
Accessibility Testing		
Internationalization Testing		
Performance Testing		
Stress Testing		
Negative Testing		
Black Box Testing		
Automated Testing		
Unit/Component Testing		
Integration Testing		

3. Разработать композицию тестов для первой поставки программного обеспечения (build 1), состоящей из трех модулей (модуль 1, модуль 2, модуль 3).

4. Разработать композицию тестов для второй поставки программного обеспечения (build 2): исправлены заведенные дефекты, доставлена новая функциональность – модуль 4.

5. Разработать композицию тестов для третьей поставки программного обеспечения (build 3): заказчик решил расширять рынки сбыта и просит осуществить поддержку программного обеспечения на английском языке.

6. Разработать композицию тестов для четвертой поставки программного обеспечения (build 4): заказчик хочет убедиться, что программное обеспечение выдержит нагрузку в 2000 пользователей.

7. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Разработанные проверки выбранного объекта реального мира для различных видов тестирования.
3. Тестовые активности для сформулированных задач.
4. Выводы по работе.

Контрольные вопросы:

1. Что такое тестирование?
2. Что такое качество программного обеспечения?
3. Что такое дефект?
4. Назовите три условия обнаружения дефекта.
5. Какие существуют виды тестирования в зависимости от объекта тестирования? Дайте характеристику каждому.
6. Какие существуют виды функционального тестирования? Дайте характеристику каждому.
7. Какие существуют виды нефункционального тестирования? Дайте характеристику каждому.
8. Какие существуют виды тестирования в зависимости от глубины покрытия? Дайте характеристику каждому.
9. Какие существуют тестовые активности? Дайте характеристику каждому.
10. Какие существуют виды тестирования в зависимости от знания кода? Дайте характеристику каждому.
11. Какие существуют виды тестирования в зависимости от степени автоматизации? Дайте характеристику каждому.
12. Какие существуют виды тестирования в зависимости от изолированности компонентов? Дайте характеристику каждому.
13. Какие существуют виды тестирования в зависимости от подготовленности? Дайте характеристику каждому.
14. Какие существуют виды тестирования в зависимости от места и времени проведения? Дайте характеристику каждому.
15. Какие этапы составляют процесс тестирования?
16. Какая композиция тестов выполняется для первой поставки программного продукта?
17. Какая композиция тестов выполняется для последующих поставок программного продукта?

Лабораторная работа №2

Разработка требований

Цель: выявить и описать пользовательские требования в виде вариантов использования (Use Cases).

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Требование (Requirement) – описание того, какие функции и с соблюдением каких условий должен выполнять программный продукт в процессе решения полезной для пользователя задачи.

Требования позволяют понять, что и с соблюдением каких условий система должна делать; предоставляют возможность оценить масштаб изменений и управлять изменениями; 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.

Работа над требованиями включает следующие этапы: выявление требований; анализ требований (моделирование бизнес-процессов, прототипирование интерфейсов, приоритизация требований, результат этапа – визуализация требований); документирование требований (результат этапа – спецификация); тестирование (валидация) требований. Работу с требованиями на этапах выявления, анализа, документирования, как правило, выполняет бизнес-аналитик. Тестирование требований выполняет тестировщик.

В иерархии требований существует три уровня: уровень бизнес-требований, уровень пользовательских требований, уровень продуктных требований (функциональные и нефункциональные требования).

Бизнес-требования выражают цель, ради которой разрабатывается продукт (зачем он нужен, какая от него ожидается польза).

Пользовательские требования описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы, и по своей сути представляют собой недетализированные функциональные требования. Поскольку здесь уже появляется описание поведения системы, требования этого уровня могут быть

использованы для оценки объема работ, стоимости проекта, времени разработки. Пользовательские требования оформляются в виде вариантов использования (Use Cases), пользовательских историй (User Stories), пользовательских сценариев (User Scenarios).

Функциональные требования описывают поведение системы, т. е. ее действия (вычисления, преобразования, проверки, обработку и т. д.). Нефункциональные требования описывают свойства системы (удобство использования, безопасность, надежность, расширяемость и т. д.), которыми она должна обладать при реализации своего поведения.

Выявление и описание требований: Use Case

Вариант использования (Use Case) продукта описывает последовательность взаимодействия системы и внешнего действующего лица. Действующим лицом может быть человек, другая система ПО или аппаратное устройство, взаимодействующее с системой для достижения некой цели.

Варианты использования меняют традиционный подход к сбору информации: пользователей не спрашивают, что, с их точки зрения, должна делать система, а выясняют, какие задачи собирается с ее помощью решать пользователь. Цель такого подхода – описать все подобные задачи. До включения каждого варианта использования в утвержденную версию требований заинтересованные в проекте лица проверяют, не выходит ли он за границы проекта. Теоретически в конечный набор вариантов использования должна входить вся желаемая функциональность системы.

Описание варианта использования включает следующие категории:

- уникальный идентификатор;
- имя, кратко описывающее задачи пользователя в формате «глагол + объект», например «разместить заказ»;
- краткое текстовое описание на естественном языке;
- список предварительных условий, которые должны быть удовлетворены до начала разработки варианта использования;
- выходные условия, описывающие состояние системы после успешного завершения разработки варианта использования;
- пронумерованный список действий, иллюстрирующий последовательность этапов взаимодействия лица и системы от предварительных условий до выходных условий.

Пример варианта использования приведен в таблице 2.1.

Таблица 2.1 – Пример варианта использования

Категории варианта использования	Описание
1	2
Идентификатор и название	UC-1 Запросить химикат
Основное действующее лицо	Сотрудник, разместивший заказ на химикат
Описание	Сотрудник, разместивший заказ на химикат, указывает в запросе необходимый химикат, вводя его название или идентификатор или импортируя его структуру из соответствующего графического средства. Система выполняет запрос, предлагая контейнер с химикатом со склада или позволяя создать запрос на заказ у поставщика
Триггер	Сотрудник указывает, что хочет заказать химикат
Предварительные условия	PRE-1 Личность пользователя аутентифицирована PRE-2 Пользователь имеет право запрашивать химикаты PRE-3 База данных по запасам химикатов доступна
Выходные условия	POST-1. Запрос сохраняется в Chemical Tracking System POST-2. Запрос отправлен на склад химикатов или поставщику
Нормальное направление развития варианта использования	1.0 Запросить химикат со склада 1.0.1 Сотрудник указывает требуемый химикат. 1.0.2 Система перечисляет контейнеры с необходимым химиктом, имеющиеся на складе. 1.0.3 Сотрудник может просмотреть историю любого контейнера. 1.0.4 Сотрудник выбирает определенный контейнер или просит отправить запрос поставщику (см. п. 1.1) 1.0.5 Сотрудник вводит остальную информацию, чтобы завершить запрос 1.0.6 Система сохраняет запрос и отправляет его на склад химикатов

Продолжение таблицы 2.1

1	2
Альтернативное направление развития варианта использования	<p>1.1 Запросить химикат у поставщика</p> <p>1.1.1 Сотрудник ищет химикат по каталогам поставщика (см. п. 1.2)</p> <p>1.1.2 Система отображает список поставщиков, где также указаны размеры, класс и цена контейнеров</p> <p>1.1.3 Сотрудник выбирает поставщика, размер, класс и количество контейнеров</p> <p>1.1.4 Сотрудник вводит остальную информацию, необходимую для запроса.</p> <p>1.1.5 Система сохраняет запрос и перенаправляет его поставщику</p>
Исключения	<p>1.2 Химиката нет в продаже</p> <p>1.2.1 Система отображает сообщение «У поставщиков нет такого химиката»</p> <p>1.2.2 Система предлагает сотруднику запросить другой химикат или выйти из программы</p> <p>1.2.3.1 Сотрудник просит запросить другой химикат (см. п. 1.2.3.2)</p> <p>1.2.4.1 Система заново начинает нормальное направление варианта использования</p> <p>1.2.3.2 Сотрудник решает выйти из системы</p> <p>1.2.4.2. Система завершает вариант использования</p>
Бизнес-правила	BR-28, BR-31

Существует несколько сценариев варианта использования (см. таблицу 2.1).

Один сценарий считается нормальным направлением развития (normal course) варианта использования, его также называют основным направлением, главным успешным сценарием и благоприятным путем. Нормальное направление для варианта использования «Запрос химиката» – запрос химиката, который есть на складе.

Другие допустимые сценарии из варианта использования называются альтернативными направлениями (alternative courses) или вторичными сценариями (secondary scenarios). Они также могут привести к успешному выполнению задания и удовлетворяют выходным условиям варианта использования. Однако они представляют вариации решения задачи или диалоговой последовательности, необходимой для выполнения задачи. В определенной точке принятия решений в диалоговой последовательности нормальное направление может перейти в альтернативное, а затем вернуться обратно в нормальное.

Условия, препятствующие успешному завершению задания, называются исключениями (exceptions). Если в процессе сбора информации не указано, как обрабатывать исключение, то возможны два пути:

- 1) разработчики предложат лучший по их мнению способ обработки исключений;
- 2) при генерации пользователем неверного условия произойдет сбой системы, т. к. никто не предусмотрел такой ситуации.

Иногда исключения рассматриваются как тип альтернативного направления, однако эти понятия следует разделять. Не обязательно реализовывать каждое альтернативное направление, которое определяют для варианта использования; кроме того, можно отложить его реализацию до следующего выпуска. Однако необходимо реализовать исключения, из-за которых завершение сценариев может оказаться неуспешным.

Расширение (extend) и включение (include)

При составлении вариантов использования часто можно столкнуться с ситуацией, когда альтернативное направление варианта использования само по себе можно выделить в автономный вариант использования. В таком случае можно расширить (extention) нормальное направление, включив этот отдельный вариант использования в нормальный поток.

Пример. Вариант использования «Запросить химикат» может включать в себя поиск по каталогу поставщика. Но при этом запросить химикат можно и без поиска по каталогам, а поиск по каталогу может выполняться как отдельная бизнес-задача пользователей. Поэтому логично расширить вариант использования «Запросить химикат» отдельным вариантом использования «Поиск по каталогам поставщика».

Иногда же несколько вариантов использования имеют общие наборы этапов. Чтобы избежать повторения этих этапов в каждом варианте использования, можно определить отдельный вариант использования и указать, что он включен (include) в другие варианты использования как подвариант.

Пример. Если покупатель совершает покупку товара, то он обязательно должен его оплатить. При этом процесс оплаты сам по себе достаточно сложный, включающий различные шаги и альтернативные варианты (оплата различными способами). Поэтому логично его выделить в отдельный вариант использования, при этом включить в вариант использования «Покупка товара».

Определение вариантов использования

Определить варианты использования можно несколькими способами:

- сначала определить действующие лица, а затем бизнес-процессы, в кото-

рых каждое лицо участвует;

- выразить бизнес-процессы в терминах определенных сценариев, обобщить сценарии в варианты использования и определить действующие лица для каждого варианта; определить внешние события, на которые система должна реагировать, а затем соотнести эти события с участвующими лицами и определенными вариантами использования;

- определить вероятные варианты использования на основе функциональных требований; если какие-либо требования невозможно проследить до какого-либо варианта использования, необходимо задуматься, нужны ли они.

Как правило, пользователи сначала определяют самые важные варианты использования, поэтому порядок предлагаемых тем позволит получить представление о приоритетах.

Преимущества применения вариантов использования состоят в том, что каждый вариант сосредоточен на поставленной задаче и пользователе. Тщательное изучение этапов взаимодействия лица и системы помогает еще на ранних стадиях разработки выявить неясности и неточности, а также позволяет составить варианты тестирования на основе вариантов использования. Способ с применением вариантов использования позволяет выявить функциональные требования, с помощью которых пользователи будут выполнять конкретные задачи. Кроме прочего, варианты использования облегчают расстановку приоритетов требований. Высшим приоритетом обладают те функциональные требования, которые созданы на основе вариантов использования с высшим приоритетом. Высший приоритет назначается по следующим причинам:

- варианты использования описывают один из основных бизнес-процессов, активизируемых системой;
- многие пользователи часто обращаются к ним;
- их запросил привилегированный класс пользователей;
- они предоставляют возможности, необходимые для соответствия требованиям;
- функции других систем зависят от их наличия.

Существуют также и преимущества технического характера. С помощью варианта использования можно выявить некоторые важные объекты предметной области и их взаимоотношения. Разработчики, использующие объектно-ориентированные методы проектирования, могут преобразовать варианты использования в объектные модели, такие как диаграммы классов и диаграммы последовательностей.

Практическое задание:

1. Получить у преподавателя задание, содержащее идею и бизнес-цели подлежащего разработке программного продукта.
2. Определить действующие лица и сформулировать наиболее вероятные варианты использования подлежащего разработке программного продукта.
3. Полностью описать три варианта использования подлежащего разработке программного продукта.
4. Для каждого варианта использования указать уникальный идентификатор; имя в формате «глагол + объект»; краткое текстовое описание; предварительные условия; выходные условия; пронумерованный список действий нормального направления развития.
5. Для каждого варианта использования при необходимости указать пронумерованный список действий альтернативного направления (направлений) развития.
6. Для каждого варианта использования при необходимости указать исключения.
7. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Описание вариантов использования подлежащего разработке программного продукта.
3. Выводы по работе.

Контрольные вопросы:

1. Что такое требование?
2. Какое значение имеют требования в проекте по разработке программного обеспечения?
3. Какие существуют этапы работы над требованиями?
4. Кто выполняет работу с требованиями?
5. Какие существуют уровни требований?
6. Что такое вариант использования?
7. Для чего нужен вариант использования?
8. Какие элементы входят в состав описания варианта использования?
9. Что такое основной сценарий варианта использования?
10. Что такое альтернативный сценарий варианта использования?
11. Что описывают в исключениях варианта использования?
12. В чем отличие альтернативного сценария от исключения в описании варианта использования?

Лабораторная работа №3

Тестирование требований

Цель: изучить критерии качества требований, выполнить тестирование требований к программному обеспечению.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Качество программного обеспечения во многом зависит от качества сформированных требований, т. к. требования к программному продукту являются базой для разработки и последующего тестирования.

Тестирование требований выполняется на предмет их соответствия критериям качества требований (рисунок 3.1).

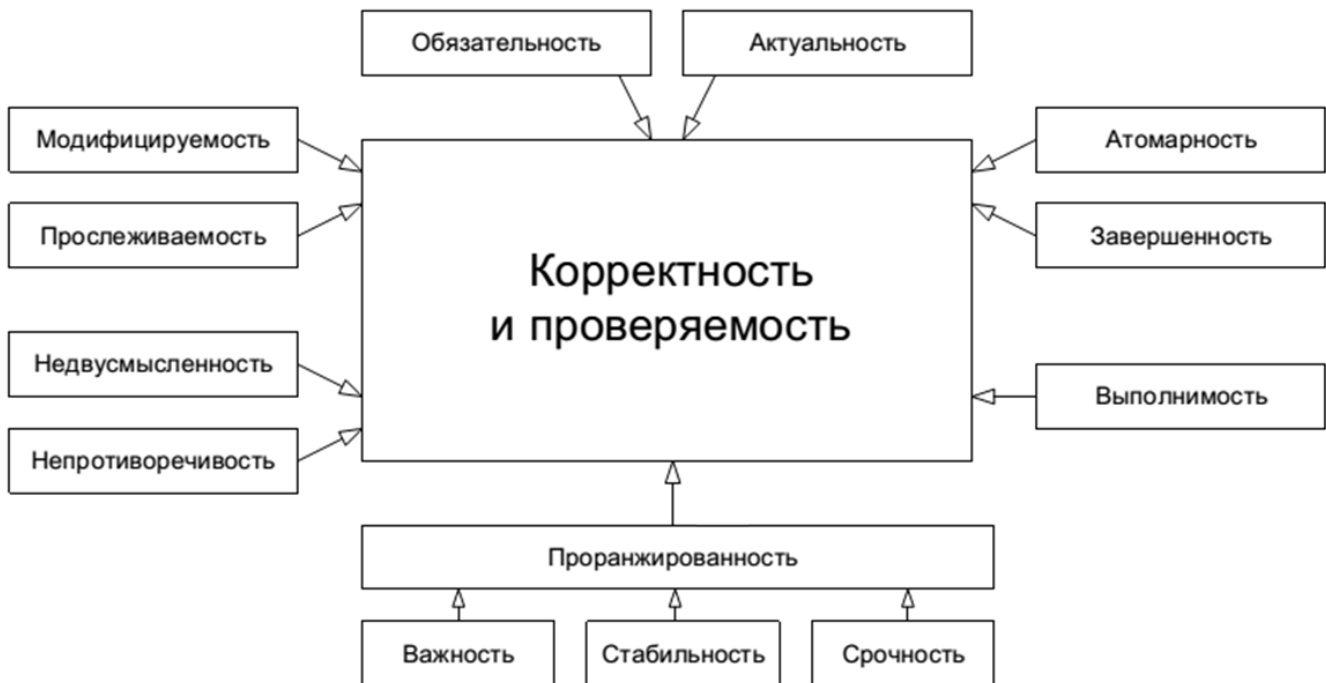


Рисунок 3.1 – Критерии качества требований

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

Типичные проблемы с завершенностью:

1. Отсутствуют нефункциональные составляющие требования или ссылки на соответствующие нефункциональные требования (например: «пароли должны храниться в зашифрованном виде», а каков алгоритм шифрования?).

2. Указана лишь часть некоторого перечисления (например: «экспорт осуществляется в форматы PDF, PNG и т. д.», а что следует понимать под «и т. д.»?).

3. Приведенные ссылки неоднозначны (например: «см. выше» вместо «см. раздел 123.45.b»).

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

Типичные проблемы с атомарностью:

1. 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 в одном предложении описаны совершенно разные элементы интерфейса в совершенно разных контекстах).

2. Требование допускает разночтение в силу грамматических особенностей языка (например: «если пользователь подтверждает 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 требования во избежание путаницы). Такое нарушение атомарности часто влечет за собой возникновение противоречивости.

3. В одном требовании объединено описание нескольких независимых ситуаций (например: «когда пользователь входит в систему, должно отображаться приветствие; когда пользователь вошел 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 более детальными требованиями).

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

Типичные проблемы с непротиворечивостью:

1. Противоречия внутри одного требования (например: «после успешного входа в систему пользователя, не имеющего права входить в систему...» – а как пользователь вошел в систему, если не имел такого права?).

2. 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 все же красной или синей?).

3. 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, 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.

Типичные проблемы с недвусмысленностью:

1. Использование терминов или фраз, допускающих субъективное толкование (например: «приложение должно поддерживать передачу больших объемов данных» – насколько «больших»?). Вот лишь небольшой перечень слов и выражений, которые можно считать верными признаками двусмысленности: адекватно, быть способным, легко, обеспечивать, как минимум, быть эффективным, своевременно, применимо, если возможно, будет определено позже, по мере необходимости, если это целесообразно, но не ограничиваясь, быть способно, иметь возможность, нормально, минимизировать, максимизировать, оптимизировать, быстро, удобно, просто, часто, обычно, большой, гибкий, устойчивый, по последнему слову техники, улучшенный, результативно.

2. 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 или какой-нибудь «Фиксатор Сообщений»?).

3. 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). Требование технологически выполнимо и может быть реализовано в рамках бюджета и сроков разработки проекта.

Типичные проблемы с выполнимостью:

1. «Озолочение» (gold plating) — требования, которые крайне долго и/или дорого реализуются и при этом практически бесполезны для конечных пользова-

телей (например: «настройка параметров для подключения к базе данных должна поддерживать распознавание символов из жестов, полученных с устройств трехмерного ввода»).

2. 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»).

3. В принципе нереализуемые требования (например: «система поиска должна заранее предусматривать все возможные варианты поисковых запросов и кэшировать их результаты»).

Обязательность, нужность (obligation) 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. Также исключены (или переработаны) должны быть требования, утратившие актуальность.

Типичные проблемы с обязательностью и актуальностью:

1. Требование было добавлено «на всякий случай», хотя реальной потребности в нем не было и нет.

2. Требованию выставлены неверные значения приоритета по критериям важности и/или срочности.

3. Требование устарело, но не было переработано или удалено.

Прослеживаемость (traceability). Прослеживаемость бывает вертикальной и горизонтальной. 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 и/или матрицы прослеживаемости.

Типичные проблемы с прослеживаемостью:

1. Требования не пронумерованы, не структурированы, не имеют оглавления, не имеют работающих перекрестных ссылок.

2. При разработке требований не были использованы инструменты и техники управления требованиями.

3. Набор требований неполный, носит обрывочный характер с явными «пробелами».

Модифицируемость (modifiability). 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 ее изменение не приводит к нарушению иных описанных в этом перечне свойств.

Типичные проблемы с модифицируемостью:

1. Требования неатомарны (см. «атомарность») и непрослеживаемы, а потому их изменение с высокой вероятностью порождает противоречивость.

2. Требования изначально противоречивы. В такой ситуации внесение изменений (не связанных с устранением противоречивости) только усугубляет ситуацию, увеличивая противоречивость и снижая прослеживаемость.

3. 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 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 ее 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) и проверяемость (verifiability). 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.

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» – мы не можем влиять на состояние пользователя).

Техники тестирования требований

1. Одной из наиболее активно используемых техник анализа требований является просмотр или рецензирование. Данная техника может быть реализована в форме:

- беглого просмотра (показ автором своей работы коллеге; самый быстрый, самый дешевый и наиболее широко используемый вид просмотра);
- технического просмотра (выполняется группой специалистов, каждый из которых представляет свою область знаний: просматриваемый продукт не может считаться достаточно качественным, пока хотя бы у одного просматривающего остаются замечания);
- формальной инспекции (структурированный, систематизированный и документируемый подход к анализу документации, для выполнения которого 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).

2. Следующей техникой тестирования и повышения качества требований является (повторное) использование такой техники выявления требований, как формулировка вопросов. Если хоть что-то в требованиях вызывает непонимание или подозрение, задавайте вопросы.

3. 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 определить, насколько требование проверяемо. Помимо использования для тестирования требований, в дальнейшем такие чек-листы и тест-кейсы могут составить основу тестовой документации.

4. Рисунки, схемы. Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы, интеллект-карты 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).

5. Исследование поведения и 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.

Практическое задание:

1. Получить у преподавателя спецификацию с требованиями к программному продукту.
2. Протестировать спецификацию методом просмотра на предмет соответствия критериям качества требований.
3. Для обнаруженных дефектов указать, какой критерий качества нарушен, и аргументировать свою точку зрения.
4. Для обнаруженных дефектов сформулировать уточняющие вопросы к заказчику для выработки качественных требований.
5. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Отчет по тестированию спецификации.
3. Выводы по работе.

Контрольные вопросы:

1. Какие выделяют критерии качества требований?
2. Какие требования являются завершенными?
3. Перечислите основные проблемы, связанные с завершенностью требований.
4. Какие требования являются атомарными?
5. Перечислите основные проблемы, связанные с атомарностью требований.
6. Какие требования являются непротиворечивыми?
7. Перечислите основные проблемы, связанные с непротиворечивостью требований.
8. Какие требования являются недвусмысленными?
9. Перечислите основные проблемы, связанные с недвусмысленностью требований.
10. Какие требования являются выполнимыми?
11. Перечислите основные проблемы, связанные с выполнимостью требова-

ний.

12. Какие требования являются обязательными, нужными и актуальными?

13. Перечислите основные проблемы, связанные с обязательностью и актуальностью требований.

14. Какие требования являются прослеживаемыми?

15. Перечислите основные проблемы, связанные с прослеживаемостью требований.

16. Какие требования являются модифицируемыми?

17. Перечислите основные проблемы, связанные с модифицируемостью требований.

18. Какие требования считаются проранжированными по важности?

19. Какие требования считаются проранжированными по стабильности?

20. Какие требования считаются проранжированными по срочности?

21. Перечислите основные проблемы, связанные с проранжированностью требований по важности.

22. Перечислите основные проблемы, связанные с проранжированностью требований по стабильности.

23. Перечислите основные проблемы, связанные с проранжированностью требований по срочности.

24. Какие требования являются корректными?

25. Перечислите основные проблемы, связанные с корректностью требований.

26. Какие существуют техники тестирования требований? В чем особенности каждой из них?

Лабораторная работа №4

Тестирование программного обеспечения: разработка тестов

Цель: разработать рабочую тестовую документацию для тестирования web-приложения.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Рабочая тестовая документация значительно улучшает качество последующего тестирования за счет анализа и детального планирования тестов. После завершения тестирования наличие тестовой документации позволяет оценить, насколько успешно были проведены все этапы тестирования, а для заказчика является подтверждением реального объема работ.

Рабочую тестовую документацию тестировщик может разрабатывать исключительно на основе спецификации еще до поставки программного обеспечения. В этом случае после поставки на тестирование версии программного продукта специалист по тестированию может сразу приступить к поиску дефектов.

Существуют следующие виды рабочей тестовой документации (таблица 4.1):

1. Check List.
2. Acceptance Sheet.
3. Test Survey.
4. Test Cases.

Основные факторы выбора тестовой документации – сложность бизнес-логики проекта, сроки проекта, размер команды и объем проекта.

На одном проекте могут комбинироваться несколько типов тестовой документации. Например, для всего проекта составлен Acceptance Sheet, но для наиболее сложных частей составлены Test Cases. Если какие-либо модули программного продукта будут подвергаться автоматизированному тестированию, то для таких модулей в обязательном порядке составляются Test Cases.

Таблица 4.1 – Виды рабочей тестовой документации и их характеристика

Тип документации	Что описывают	Когда используют
Check List	Вспомогательный тип документации, содержащий список основных проверок	Для типовой функциональности
Acceptance Sheet	Перечень всех модулей и функций приложения, подлежащих проверке	Небольшие (до 3 месяцев), простые по бизнес-логике проекты
Test Survey	Перечень всех модулей и функций приложения, а также конкретные проверки для них. Может содержать ожидаемый результат	Средние или большие проекты с понятной бизнес-логикой
Test Cases	Перечень всех модулей и функций приложения, а также конкретные проверки для них. Каждая проверка содержит набор входных значений, предусловий, шагов выполнения и ожидаемых результатов. Всегда приводятся ожидаемые результаты для каждого шага проверки	Большие и долгосрочные проекты, проекты со сложной бизнес-логикой, проекты с большой командой

При составлении рабочей тестовой документации необходимо указать номер тестируемой сборки, тип выполняемой тестовой активности, период времени тестирования, ФИО тестировщика, тестовое окружение (операционная система, браузер и др.).

Рабочая тестовая документация представляет собой перечень всех проверок для модулей/подмодулей приложения. В качестве одного модуля, как правило, выступает рабочее окно приложения, в качестве подмодулей – логически завершённые блоки этого окна.

Для каждого модуля в обязательном порядке выполняется тестирование GUI, а также общие функциональные проверки (General) для любого типа приложения (навигация, скроллинг, табуляция и др.). Далее в рамках модуля формулируются проверки соответствия функционала приложения требованиям, заявленным в спецификации. Степень детализации каждой из таких функциональных проверок зависит от выбранного типа тестовой документации (Acceptance Sheet, Test Survey, Test Cases). В частности, для Acceptance Sheet все проверки только перечисляют. Для Test Survey для каждого элемента прописывают позитивные и негативные

проверки, источником которых могут служить базовые проверки (в виде чеклиста) для соответствующих элементов GUI. Для Test Cases каждую из позитивных и негативных проверок описывают в виде последовательности шагов с указанием ожидаемого результата. Для Test Survey напротив каждой проверки указывается глубина тестирования: Smoke, MAT, AT. Для Acceptance Sheet в качестве глубины тестирования всегда указывается AT.

Примеры фрагментов рабочей тестовой документации Check List, Acceptance Sheet, Test Survey приведены в таблице 4.2.

Таблица 4.2 – Примеры Check List, Acceptance Sheet, Test Survey

Вид рабочей тестовой документации	Пример	Глубина тестирования
1	2	3
Check List	Протестировать форму авторизации	–
Acceptance Sheet	Форма авторизации: 1. GUI. 2. General. 3. Поле «Эл. адрес». 4. Поле «Пароль». 5. Кнопка «Войти». 6. Чекбокс «Не выходить из системы». 7. Ссылка «Забыли пароль»	AT AT AT AT AT AT
Test Survey	Форма авторизации: 1. GUI. 2. General. 3. Валидный эл. адрес + валидный пароль. 4. Валидный эл. адрес с пробелами в начале и в конце + валидный пароль с пробелами в начале и в конце. 5. Валидный эл. адрес с пробелами в середине + валидный пароль с пробелами в середине. 6. Валидный эл. адрес минимально допустимой длины + валидный пароль минимально допустимой длины. 7. Валидный эл. адрес максимально допустимой длины + валидный пароль максимально допустимой длины. 8. Пустое поле эл. адреса + пустое поле пароля.	AT AT Smoke MAT MAT MAT MAT AT

Продолжение таблицы 4.2

1	2	3
	9. Валидный эл. адрес + невалидный пароль (специальные символы).	АТ
	10. Валидный эл. адрес + невалидный пароль (русские буквы).	АТ
	11. Валидный эл. адрес + невалидный пароль (длина превышает максимально допустимую).	АТ
	12. Валидный эл. адрес + невалидный пароль (длина меньше минимально допустимой).	АТ
	13. Невалидный эл. адрес (специальные символы кроме '_', '-', '@', '.', ') + валидный пароль.	АТ
	14. Невалидный эл. адрес (русские буквы) + валидный пароль.	АТ
	15. Невалидный эл. адрес (использование символа '@' дважды) + валидный пароль.	АТ
	16. Невалидный эл. адрес (отсутствие символа '.') + валидный пароль.	АТ
	17. Невалидный эл. адрес (длина превышает максимально допустимую) + валидный пароль.	АТ
	18. Невалидный эл. адрес (длина меньше минимально допустимой) + валидный пароль.	АТ
	19. Запомнить данные: выйти из системы и зайти обратно.	МАТ
	20. Ссылка «Забыли пароль»	МАТ

Тест-кейс (таблица 4.3) – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Под тест-кейсом также понимают соответствующий документ, представляющий формальную запись тест-кейса.

Высокоуровневый тест-кейс – тест-кейс без конкретных входных данных и ожидаемых результатов. Как правило, ограничивается общими идеями и операциями, схож по своей сути с подробно описанными пунктами чек-листа.

Низкоуровневый тест-кейс – тест-кейс с конкретными входными данными и ожидаемыми результатами. Представляет собой полностью готовый к выполнению тест-кейс и 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.

В зависимости от внутренних шаблонов компании и инструмента управления тест-кейсами внешний вид их записи может немного отличаться (могут быть добавлены или убраны отдельные поля), но концепция остается неизменной.

Общая структура тест-кейса включает (см. таблицу 4.3):

- идентификатор;
- связанное с тест-кейсом требование;
- модуль и подмодуль приложения;
- заглавие тест-кейса;
- исходные данные, приготовления к тест-кейсу;
- шаги тест-кейса;
- ожидаемые результаты по каждому шагу тест-кейса.

Таблица 4.3 – Пример Test Case

Номер тест-кейса	Приоритет	Требование	Модуль	Подмодуль	Описание тест-кейса	Ожидаемые результаты
18	A	REQ3.7	Главная страница	Форма авторизации	Авторизация с помощью валидного эл. адреса и валидного пароля. 1. Ввести в поле «Эл. адрес» abc@mail.ru. 2. Ввести в поле «Пароль» qwerty. Нажать кнопку «Войти»	1. В поле «Эл. адрес» отображается abc@mail.ru. 2. В поле «Пароль» отображается qwerty. Осуществляется переход на домашнюю страницу пользователя abc@mail.ru, qwerty

Идентификатор 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 включать префиксы, суффиксы и иные осмысленные

компоненты, позволяющие быстро определить цель тест-кейса и часть приложения (или требований), к которой он относится.

Приоритет показывает важность тест-кейса. Он может быть выражен буквами (А, В, С, 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 трех до пяти. Приоритет тест-кейса может коррелировать с важностью требования, с которым связан тест-кейс; потенциальной важностью дефекта, на поиск которого направлен тест-кейс. Основная задача этого атрибута – упрощение распределения внимания и усилий команды, 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 показывает то основное требование, проверке выполнения которого посвящен тест-кейс. Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость. При этом следует отметить, что некоторые тест-кейсы могут разрабатываться вне прямой привязки к требованиям. Хотя такой вариант и не считается хорошим, он достаточно распространен.

Модуль и подмодуль приложения 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. Тогда приложение логически разделяется на компоненты (модули), а те, в свою очередь, на более мелкие компоненты (подмодули). Для таких небольших частей приложения разрабатывать тест-кейсы становится намного проще. Как правило, иерархия модулей и подмодулей создается 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. 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.

Заглавие тест-кейса 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. Заглавие тест-кейса должно быть информативным, уникальным.

Исходные данные, необходимые для выполнения тест-кейса, позволяют описать все то, что должно быть подготовлено до начала выполнения тест-кейса,

например, состояние базы данных, состояние файловой системы и ее объектов, состояние серверов и сетевой инфраструктуры.

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

- не пишите лишних начальных шагов (запуск приложения, очевидные операции с интерфейсом и т. п.);
- даже если в тест-кейсе всего один шаг, нумеруйте его;
- 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 четко прописанных значений и указаний;
- ссылайтесь на предыдущие шаги и их диапазоны для сокращения объема текста (например, «повторить шаги 3–5 со значением...»);
- пишите шаги последовательно, без условных конструкций вида «если... то...».

Ожидаемые результаты 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, вместо недопустимого «приложение работает верно» следует писать «появляется окно с надписью...»);
- пишите ожидаемый результат по всем шагам без исключения, даже если результат некоего шага будет совершенно тривиальным и очевидным;
- пишите кратко, но не в ущерб информативности;
- избегайте условных конструкций вида «если... то...».

Наличие тест-кейсов позволяет:

- структурировать и систематизировать подход к тестированию;
- вычислять метрики тестового покрытия и принимать меры по его увеличению;
- отслеживать соответствие текущей ситуации плану (сколько примерно понадобится тест-кейсов, сколько уже есть, сколько выполнено 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;
- проводить регрессионное тестирование и повторное тестирование;
- повышать качество требований.

Далее рассмотрим подробно базовые проверки графического интерфейса пользователя и функциональности web-, desktop- и mobile-приложений.

Для любого приложения выполняется тестирование графического интерфейса пользователя (таблица 4.4).

Таблица 4.4 – Перечень основных GUI-проверок для всего приложения

Название проверки	Описание проверки
1	2
1. Правописание	Лексические, грамматические и пунктуационные ошибки
2. Расположение и выравнивание	Выравнивание по левому или правому краю (в зависимости от требований приложения), отступы, идентичность расстояний между названием и полем. Корректное расположение текста, длинный текст не выходит за границы поля при вводе
3. Длинные названия	Длинные названия корректно обрезаются с помощью многоточия в конце, при наведении возникают хинты с полнотекстовым вариантом
4. Соответствие названий форм/элементов GUI их назначению	Проверка названий форм/элементов GUI с точки зрения их смысловой нагрузки
5. Унификация (стиля, цвета, шрифта, названий)	Единообразие цвета, шрифта, размеров (высоты/ширины), выравнивания полей, названий полей, категорий меню и др. в рамках всего приложения
6. Эффект «нажатия»	Изменение вида ссылок, кнопок, позиций меню и др. при наведении курсора. Изменение вида курсора при наведении на ссылки, кнопки, позиции меню и др.
7. Хинты	Проверка всплывающих подсказок с точки зрения правописания, выравнивания, соответствия назначению

Продолжение таблицы 4.4

1	2
8. Сообщения об успешном/неуспешном завершении действия, о подтверждении действия	<p>Проверка верхней панели (логотипа и названия) формы с сообщением.</p> <p>Если присутствует кнопка «Отмена», то в правом верхнем углу формы с сообщением присутствует «крестик» для альтернативной возможности закрыть форму.</p> <p>Сообщения о подтверждении удаления по умолчанию активированы на кнопку «нет»</p>
9. Изменение размеров окна, изменение масштаба страницы	<p>Появление скроллинга при уменьшении размера окна.</p> <p>Сохранение взаимного расположения элементов при уменьшении окна, изменении масштаба).</p> <p>Перераспределение элементов с сохранением пропорций при изменении масштаба страницы</p>

Общие проверки функциональности для всех типов приложений, а также общие проверки для web-приложений приведены в таблицах 4.5, 4.6.

Таблица 4.5 – Перечень общих проверок функциональности для любого типа приложения

Название проверки	Описание проверки
1. Табуляция	Перемещение с помощью клавиатуры должно осуществляться сверху вниз слева направо. Недоступные поля должны пропускаться
2. «Хлебные крошки»	«Хлебные крошки» – элемент навигации, являющийся признаком удобства пользования приложением в целом и перемещением по его структуре
3. Скроллинг	<p>Отсутствие скроллинга в случае, если текст вмещается на странице без прокрутки.</p> <p>Соответствующее изменение текста при использовании скроллинга.</p> <p>Возможность изменения положения скроллинга при помощи мыши, кнопок Page up/down, Home/End</p>
4. Взаимосвязь компонентов	Поведение одного компонента при изменении/удалении другого (например, при удалении категории товара не должны удаляться все товары в этой категории)
5. Фокус на кнопке для исполнения действий	Ввод данных → нажатие Enter → действие осуществилось

Таблица 4.6 – Перечень общих проверок функциональности для web-приложений

Название проверки	Описание проверки
1. Подготовка к тестированию	Перед тестированием каждой новой сборки необходимо осуществить очистку кэша и cookies. Для этого можно воспользоваться приложением CCleaner
2. 404 Error	Переход по некорректному адресу должен вести на страницу с Error 404, а не на страницу Page cannot be found, например. Страница Error 404 должна быть реализована в общем дизайне тестируемого приложения
3. Логотип	Логотип должен быть ссылкой на главную страницу
4. Email нотификации	Проверка работоспособности отправки email-нотификаций (как администратору, так и пользователю), если только отсутствие писем не является спецификой проекта
5. Отображение flash-элементов при отключенном или не установленном в браузере flash-плеере	Пользователю должно быть предложено скачать и установить последнюю версию flash-плеера; на месте flash-объекта должно отображаться альтернативное изображение
6. Проверка работоспособности приложения при отключенном JavaScript	Основная функциональность и навигация должна работать

В таблицах 4.7–4.16 приведены основные проверки для элементов пользовательского интерфейса: поле для ввода данных, поле для загрузки файлов, поле для ввода даты, поле со списком/выпадающий список, кнопка, радиобаттон, чекбокс, меню, таблица, календарь, ссылка, сообщения, поп-ап (всплывающие окна).

Таблица 4.7 – Перечень основных проверок для поля ввода данных

Functional Test	GUI Test
<ol style="list-style-type: none"> 1. Обязательность ввода. 2. Обработка только пробелов. 3. Использование пробелов в тексте: <ol style="list-style-type: none"> 3.1. Пробелы в начале и в конце строки должны отсекаются при сохранении. 3.2. Пробелы внутри текста отсекаются не должны. 4. Минимально/максимально допустимое количество символов. 5. Формат данных (исходя из его логического назначения и требований приложения). 6. Формат числовых данных (если допускаются): негативные, дробные с точкой и запятой. 7. Использование специальных символов (введенные символы должны отобразиться в том же виде, в котором они были введены, если только ввод специальных символов не запрещен требованиями приложения). 8. Возможность редактирования введенных значений. 9. Корректное распределение текста по строкам (переход на новую строку автоматически). 10. Уникальные данные (например, уникальность логина, email). 11. Автоматическая постановка курсора в первое поле для ввода при открытии формы. 12. Ввод тегов и скриптов (введенные теги и скрипты должны отобразиться в том же виде, в котором они были введены). 	<ol style="list-style-type: none"> 1. Название поля (правописание, соответствие названия тематике модуля/страницы). 2. Выравнивание названий полей (выравнивание по левому или правому краю в зависимости от требований приложения, отступы, идентичность расстояний между названием и полем). 3. Корректное расположение текста, длинный текст не выходит за границы поля при вводе. 4. Унификация дизайна по отношению ко всему приложению (цвет, шрифт, размер (высота/ширина), выравнивание полей). 5. Расположение вводимого текста внутри поля (унификация, выравнивание)

Таблица 4.8 – Перечень основных проверок для поля загрузки файлов

Functional Test	GUI Test
1	2
<ol style="list-style-type: none"> 1. Обязательность выбора файла. 2. Форматы: корректные/некорректные. 3. Корректный формат, но отсутствует/модифицировано расширение. 	<ol style="list-style-type: none"> 1. Унификация дизайна по отношению ко всему приложению (цвет, шрифт, высота/ширина).

Продолжение таблицы 4.8

1	2
<p>4. Ограничения на размер (включая загрузку файлов нулевого размера, большого размера). Загрузка исполняемых файлов (EXE, PHP, JSP др.).</p> <p>5. Загрузка переименованного EXE-файла.</p> <p>6. Путь к файлу меньше 259 символов.</p> <p>7. Путь к файлу равен 260 символов.</p> <p>8. Путь к файлу больше 260 символов.</p> <p>9. Корректный путь введен с клавиатуры.</p> <p>10. Имитировать сбой загрузки (например, с использованием flash-накопителя).</p> <p>11. Одновременная загрузка нескольких файлов</p>	<p>2. Выравнивание названий загруженных файлов.</p>

Таблица 4.9 – Перечень основных проверок для радиобаттона

Functional Test	GUI Test
<p>1. Функциональность: включение/выключение.</p> <p>2. Не может быть меньше двух радиокнопок.</p> <p>3. По умолчанию одна радиокнопка должна быть включена.</p> <p>4. Не может быть включено более одной радиокнопки.</p> <p>5. При переходе на следующую страницу и возвращении назад выбранная радиокнопка не должна сбрасываться.</p> <p>6. Активация путем нажатия как на символ, так и на текст</p>	<p>1. Унификация дизайна по отношению ко всему приложению.</p> <p>2. Выравнивание расположения радиобаттона с соответствующим названием.</p> <p>3. Выравнивание расположений радиобаттонов.</p> <p>4. Изменение радиобаттона при наведении курсора.</p> <p>5. Изменение курсора при наведении на радиобаттон</p>

Таблица 4.10 – Перечень основных проверок для чекбоксов

Functional Test	GUI Test
1	2
<p>1. Функциональность: включение/выключение.</p> <p>2. Наличие дополнительного чекбокса, выставляющего/снимающего все чекбоксы при наличии больше 10 чекбоксов.</p>	<p>1. Унификация дизайна по отношению ко всему приложению.</p> <p>2. Выравнивание расположения чекбокса с соответствующим названием.</p>

Продолжение таблицы 4.10

1	2
<p>3. При переходе на следующую страницу и возвращении назад выбранный чекбокс не должен сбрасываться.</p> <p>4. Активация путем нажатия как на символ, так и на текст</p>	<p>3. Изменение чекбокса при наведении курсора.</p> <p>4. Изменение курсора при наведении на чекбокс</p>

Таблица 4.11 – Перечень основных проверок для поля со списком

Functional Test	GUI Test
<p>1. Сортировка по алфавиту или по смыслу.</p> <p>2. В случае, если значения выходят за границы списка и нет возможности увеличения размера списка, то необходимо отображение хинтов (всплывающих подсказок).</p> <p>3. Выбор пункта списка по нажатию соответствующей первой буквы на клавиатуре.</p> <p>4. Возможность введения значений вручную (если это позволяет приложение).</p> <p>5. Возможность выбора значения из списка как с помощью мыши, так и с клавиатуры</p>	<p>1. Правописание.</p> <p>2. Подсветка при выборе каждого из значений, при выборе нескольких значений одновременно.</p> <p>3. Унификация дизайна (цвет, шрифт, размер (высота/ширина), цвет подсветки, выравнивание)</p>

Таблица 4.12 – Перечень основных проверок для меню

Functional Test	GUI Test
<p>1. Осуществление соответствующего перехода при выборе пункта меню.</p> <p>2. Визуальное различие в момент работы на определенной вкладке (подсветка, подчеркивание)</p>	<p>1. Подсветка категории меню при наведении курсора.</p> <p>2. Изменение курсора при наведении на категорию меню.</p> <p>3. Если в данный момент выполняется работа в выбранной вкладке, то в меню она отличается визуально и является неактивной.</p> <p>4. Совпадение названий категорий меню в случае, если меню дублируется в нескольких местах</p>

Таблица 4.13 – Перечень основных проверок для ссылки

Functional Test	GUI Test
<ol style="list-style-type: none"> 1. Функционирование ссылки (должен осуществиться переход на соответствующую страницу). 2. Переход по загруженной ссылке должен осуществляться в новой вкладке или во всплывающем окне. 3. Форматы ссылок и префиксов. 4. Срабатывание ссылки только при нажатии на саму ссылку, а не на пустую область возле нее 	<ol style="list-style-type: none"> 1. Унификация стилей (в соответствии с дизайном сайта). 2. Расположение ссылок (в соответствии с дизайном сайта). Например, расположение всех ссылок слева или справа от элементов. 3. Названия (унификация, идентичность названий ссылок одинакового назначения, спеллинг, соответствие с открытым модулем или страницей, вместимость названия ссылки в отведенном блоке). 4. Изменение вида курсора при наведении на ссылку. 5. Изменение вида ссылки при наведении курсора (подчеркивание)

Таблица 4.14 – Перечень основных проверок для таблицы

Functional Test	GUI Test
<ol style="list-style-type: none"> 1. При появлении нескольких страниц есть кнопки Вперед, Назад, На первую, На последнюю страницу (пагинация). 2. Проверка сортировок, в том числе сортировки по умолчанию. 3. Обновление значений таблицы после добавления, изменения, удаления данных. 4. Единичное/множественное выделение нескольких значений 	<ol style="list-style-type: none"> 1. Унификация дизайна для всего приложения (цвет, шрифт, размер (высота/ширина), выравнивание). 2. Название (соответствие с текущим модулем, спеллинг). 3. Выравнивание значков сортировки в названии колонок. 4. Выравнивание названий колонок, значений внутри таблицы. 5. Корректное отображение длинных названий (соответствующие переходы на новые строки, сокращение названий (появление многоточия либо сокращение по слову)). 6. Корректное отображение данных после использования сортировки (размеры колонок и столбцов фиксированы, текст не разбивает структуру таблицы)

Таблица 4.15 – Перечень основных проверок для календаря

Functional Test	GUI Test
<ol style="list-style-type: none"> 1. Ввод даты с помощью календаря. 2. Ввод даты вручную: разные форматы, номер месяца: > 12, день: > 31 (+ для февраля). 3. Логика работы поля (например, подсчет возраста после ввода даты рождения; невозможность ввести дату рождения свыше текущего дня) 	<ol style="list-style-type: none"> 1. Унификация дизайна для всего приложения (цвет, шрифт, размер (высота/ширина), выравнивание). 2. Отображение календаря рядом с полем. 3. Корректное выравнивание всех элементов и ссылок в календаре

Таблица 4.16 – Перечень основных проверок для сообщений

Functional Test	GUI Test
<ol style="list-style-type: none"> 1. Пользователь должен быть информирован о действиях, происходящих в системе посредством сообщений об успешном завершении операции. 2. На необратимые действия, такие как удаление, должны быть подтверждающие сообщения. 3. Введенные в форму данные не должны сбрасываться после появления сообщения 	<ol style="list-style-type: none"> 1. Правописание сообщений. 2. Соответствие сообщений смыслу выполняемого действия. 3. Соответствие названий полей в сообщениях названиям полей, форм, таблиц, кнопок и т. д. 4. Унификация стилей (цвет, размер) для всего приложения. 5. Если присутствует кнопка «Отмена», то в правом верхнем углу формы с сообщением присутствует «крестик» для альтернативной возможности закрыть форму. 6. Сообщения о подтверждении удаления по умолчанию активированы на кнопку «нет». 7. Соответствие цвета типу сообщения (красный для сообщений об ошибках, зеленый для сообщений об успешном завершении операции). 8. Невалидное значение не должно отображаться в сообщении об ошибке (неправильно: «Email 2309234@@mail.ru не соответствует допустимому формату»). 9. Согласование числительного и связанного с ним существительного (например, «1 день», «2 дня»)

Практическое задание:

1. Получить у преподавателя спецификацию с требованиями к web-приложению.
2. В зависимости от сложности бизнес-логики web-приложения выбрать наиболее подходящий вид рабочей тестовой документации (Acceptance Sheet, Test Survey, Test Cases).
3. Анализировать web-приложение разбить на модули и подмодули.
4. Разработать рабочую тестовую документацию для всех модулей и подмодулей web-приложения.
5. Указать номер тестируемой сборки, название приложения, тип выполняемой тестовой активности, период времени тестирования, ФИО тестировщика, тестовое окружение (операционная система, браузер).
6. Предусмотреть проверки GUI для каждого модуля.
7. Предусмотреть общие функциональные проверки (General) для каждого модуля.
8. В рамках каждого модуля предусмотреть функциональные проверки. Степень детализации каждой из функциональных проверок должна соответствовать выбранному на этапе 1 типу тестовой документации.
9. Для каждой проверки указать глубину тестового покрытия (Smoke, MAT, AT) с учетом выбранного на этапе 1 типа тестовой документации.
10. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Рабочая тестовая документация.
3. Выводы по работе.

Контрольные вопросы:

1. Какие существуют разновидности рабочей тестовой документации?
2. Check List: что описывают и когда используют?
3. Acceptance Sheet: что описывают и когда используют?
4. Test Survey: что описывают и когда используют?
5. От чего зависит степень детализации каждой функциональной проверки?
6. Какая глубина тестирования указывается для проверок в Acceptance Sheet?
7. Какая глубина тестирования указывается для проверок в Test Survey?
8. Что такое Test Case?
9. Какова структура описания Test Case?
10. Что содержит Идентификатор в описании Test Case?

11. Что приводится в поле Приоритет описания Test Case?
12. Что приводится в поле Требование описания Test Case?
13. Что приводится в поле Модуль и подмодуль приложения описания Test Case?
14. Что приводится в поле Заглавие описания Test Case?
15. Что приводится в поле Исходные данные, приготовления описания Test Case?
16. Что приводится в поле Шаги описания Test Case?
17. Что приводится в поле Ожидаемые результаты описания Test Case?
18. Для чего нужны Test Cases?
19. Какие проверки выполняют при тестировании GUI?
20. Какие общие функциональные проверки выполняют для всего приложения?
21. Перечислите базовые проверки для поля ввода данных.
22. Перечислите базовые проверки для поля загрузки файлов.
23. Перечислите базовые проверки для ввода даты.
24. Перечислите базовые проверки для поля со списком.
25. Перечислите базовые проверки для радиобаттона.
26. Перечислите базовые проверки для чекбокса.
27. Перечислите базовые проверки для меню.
28. Перечислите базовые проверки для таблиц.
29. Перечислите базовые проверки для ссылок.
30. Перечислите базовые проверки для сообщений.

Лабораторная работа №5

Поиск и документирование дефектов

Цель: протестировать web-приложение и описать найденные дефекты.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Дефекты, обнаруженные тестировщиком, должны быть корректно и понятно описаны, чтобы разработчик смог воспроизвести данный дефект и устранить его. Описание каждого дефекта сохраняется в специализированной – багтрекинговой – системе (например, JIRA, Bugzilla, Mantis, Redmine и др.) или в предварительно созданном в программной среде Microsoft Excel файле (пример приведен в таблице 5.1).

Таблица 5.1 – Пример описания дефекта

№	Название дефекта	Важность	Алгоритм воспроизведения	Фактический результат	Ожидаемый результат	Приложение	Примечание
39	Администраторская часть: Файлы: Выбор файла: Ссылка «Скачать файл»: Администратор не имеет возможности скачать загруженные студентом файлы	Critical	Шаги по воспроизведению: 1. Входим на веб-сайт ... 2. Проходим авторизацию: admin/admin. 3. Выбираем в меню категорию «Файлы». 4. Выбираем подкатегорию меню «Выбор файла». 5. Выбираем в поле «Обзор файла» любой доступный файл. 6. Нажимаем на ссылку «Скачать файл»	Переход к странице «HTTP Status 404»	Происходит скачивание выбранного файла	39.png	REQ-26

Описание дефекта включает следующие обязательные поля:

1. **Headline** – название дефекта.
2. **Severity** – степень критичности (важность дефекта).
3. **Description** – алгоритм воспроизведения.
4. **Result** – фактический результат.
5. **Expected result** – ожидаемый результат.
6. **Attachment** – прикрепленные файлы (приложение).

В багтрэкинг-системах для каждого дефекта автоматически генерируется его уникальный номер, в случае использования Microsoft Excel номер дефекту необходимо присваивать вручную.

Требование спецификации, которое нарушает обнаруженный дефект, можно дополнительно вынести в примечание.

Дополнительно в описании дефекта может быть указана **Priority** – степень срочности исправления дефекта разработчиком.

Рассмотрим подробно каждую категорию описания дефекта.

Headline (название дефекта). Цель составления заголовка дефекта – представить краткую и в тоже время понятную информацию о том, где, что и в результате чего произошло. Характеристиками качественного заголовка являются краткость, информативность, точная идентификация проблемы.

Заголовок дефекта должен отвечать на три вопроса:

1. Где? В каком месте интерфейса пользователя находится проблема.

В данной части заголовка следует также дополнительно указать особенности теста, если это поможет разобраться в проблеме (версия операционной системы, браузера, сторонних приложений, которые имеют отношение к тестируемому программному средству).

2. Что? Что происходит или не происходит согласно спецификации или представлению о нормальной работе программного продукта. При этом необходимо указывать на наличие проблемы, а не на ее содержание (его указывают в описании). Если содержание проблемы варьируется, все известные варианты указываются в описании.

3. Когда (при каких условиях)? В какой момент работы программы или по наступлению какого события проблема проявляется.

Пример: в приложении есть диалог «Преобразовать данные» с кнопкой «Преобразовать». При нажатии этой кнопки появляется сообщение об ошибке «Ошибка 315». Заголовок дефекта по описанной методике составляется так:

Где?: Диалог «Преобразовать данные».

Что?: Показывается сообщение об ошибке.

Когда?: При нажатии кнопки «Преобразовать».

Итоговый заголовок будет иметь следующий вид: Диалог «Преобразовать данные» показывает сообщение об ошибке при нажатии кнопки «Преобразовать». Уберем лишние слова, добавим код ошибки для удобства поиска: Диалог «Преобразовать данные»: сообщение об ошибке 315 при нажатии кнопки «Преобразовать».

Если сформулировать заголовок по формуле Где? Что? Когда (при каких условиях)? трудно, то можно воспользоваться следующим алгоритмом действий:

1. Пропустите заголовок дефекта.
2. Напишите описание дефекта, фактический и ожидаемый результаты.
3. Выделите ключевые моменты, руководствуясь формулой Где? Что? Когда (при каких условиях)?
4. Сложите эти ключевые моменты вместе.
5. То, что получится в итоге, и будет составлять заголовок дефекта.

Severity (степень критичности). Степень критичности (серьезности, важности) показывает степень ущерба, который наносится проекту существованием дефекта.

В общем случае выделяют следующие градации критичности дефектов (таблица 5.2): Critical (критический), Major (значительный), Average (средней значимости), Minor (незначительный), Enhancement (предложение по улучшению).

Description (алгоритм воспроизведения). Цель составления алгоритма воспроизведения дефекта – последовательно описать шаги для повторения дефекта. Description должен быть оформлен в виде списка перечисления действий:

1. Шаг #1.
2. Шаг #2.
- ...
- n. Шаг #n.

В случае, если для воспроизведения дефекта требуется ряд начальных условий (например, должен быть создан определенный набор документов, пользователь или группа пользователей с особыми правами), то эти предусловия должны быть вынесены в начало описания:

- Предусловия.
1. Шаг #1.
 2. Шаг #2.
 - ...
 - n. Шаг #n.

Таблица 5.2 – Степени критичности дефектов

Severity	Описание	Примеры
Critical (критический)	<p>Функциональная ошибка, которая блокирует работу части функционала или всего приложения.</p> <p>Функциональная ошибка, которая нарушает ключевую (с точки зрения конечного пользователя или бизнеса заказчика) функциональность приложения</p>	<p>Заблокирована вкладка «Категория меню».</p> <p>Неправильно подсчитывается итоговая сумма при вводе скидочного промокода.</p> <p>Раскрывается конфиденциальная информация</p>
Major (значительный)	<p>Функциональная ошибка, которая нарушает нормальную работу приложения, но не блокирует работу части функционала в целом</p>	<p>Невозможно загрузить видеофайлы на персональной страничке.</p> <p>Не работает система email-нотификации.</p> <p>Не работает интеграция с социальными сетями.</p> <p>Необходимо перезапускать приложение при выполнении типичных сценариев работы</p>
Average (средней значимости)	<p>Не очень важная функциональная ошибка.</p> <p>Критичные дефекты GUI</p>	<p>Не работает сортировка.</p> <p>«Уехал» текст за пределы окна</p>
Minor (незначительный)	<p>Редко встречающиеся незначительные функциональные дефекты.</p> <p>90 % дефектов GUI</p>	<p>Введены необязательные для заполнения поля, которых нет в спецификации.</p> <p>Грамматические, пунктуационные ошибки</p>
Enhancement (предложение по улучшению)	<p>Функциональные предложения, советы по улучшению дизайна (оформления), навигации и др.</p> <p>Такой дефект является необязательным для исправления</p>	<p>Добавить кнопку «Наверх» на длинных формах.</p> <p>Увеличить размер шрифта</p>

При составлении Description необходимо следовать приведенным ниже рекомендациям.

Description – это четкий алгоритм, в котором приветствуются короткие, понятные фразы и нумерация.

Нельзя использовать личные предложения формата «Я думаю, что так будет лучше», «Я зашел на страницу...» и т. д.

Можно использовать специальные символы «+», «=», «<>», которые помогут сделать подобие навигации: File > Open, DOC + XLS. Однако не рекомендуется писать шаги в строку через символы перехода – это затрудняет восприятие дефекта.

Следует указывать специфичные условия воспроизведения дефекта, если таковые имеются. Например, под каким пользователем вы работаете (если это важно).

Описание предложений по улучшению должно быть максимально полным и аргументированным.

Result (фактический результат). Цель написания Result – четко описать полученный результат.

Expected result (ожидаемый результат). Цель написания Expected result – привести аргументы разработчикам, как именно должно работать приложение.

В Expected result должно быть четкое обоснование, почему именно так должно работать приложение. Лучше всего, если в нем приведена ссылка на конкретный пункт спецификации.

Если в Expected result приводится ссылка на спецификацию, сам тестировщик дополнительно цитирует текст спецификации, чтобы сократить время разработчика на анализ документа и однозначно указать на способ решения проблемы.

Если функция работает, но некорректно, то в Expected Result обязательно должно быть описание того, как она должна работать корректно.

Если сделана ошибка в надписи или интерфейсе проекта, необходимо грамотно и полностью указать, как она должна быть исправлена – написать надпись без ошибки или описать требуемые изменения интерфейса.

Expected Result всегда следует заполнять. Не стоит полагаться на очевидность представлений о правильном поведении приложения.

Текст Expected result рекомендуется писать законченными полными безличными предложениями.

Attachment (приложения). Attachment – это прикрепленный к дефекту файл, дополняющий описание: скриншот, файлы, необходимые для воспроизведения дефекта, логи программы, видео ошибки и т. д. Attachment является вспомогатель-

ным средством передачи информации о проблеме. Для всех GUI дефектов attachment обязателен.

Если к дефекту прикрепляется файл, об этом обязательно должно быть указано в описании дефекта («See the file/screenshot/log/video attached»). Прикрепленный файл не должен быть слишком большим по размеру (особенно это касается видео: до 10 Мбайт), а также должен относиться именно к описанной ошибке (например, из лога приложения стоит скопировать в прикрепляемый файл только данные об ошибке). Формат файла скриншота – PNG. Имя файла скриншота рекомендуется делать числовым, нейтральным – 1.png, 25.png и т. п.

Скриншот должен содержать следующие элементы: сама ошибка, выделение места ошибки, стрелка к прямоугольнику, описание ошибки: «Наблюдаемый результат» и/или «Ожидаемый результат». Текст на скриншоте также необходимо выделить: обвести в прямоугольник и набрать шрифтом, заметно отличающимся от шрифта программы. Качественно подготовленный скриншот должен давать возможность понять смысл дефекта без необходимости читать его описание (рисунок 5.1).

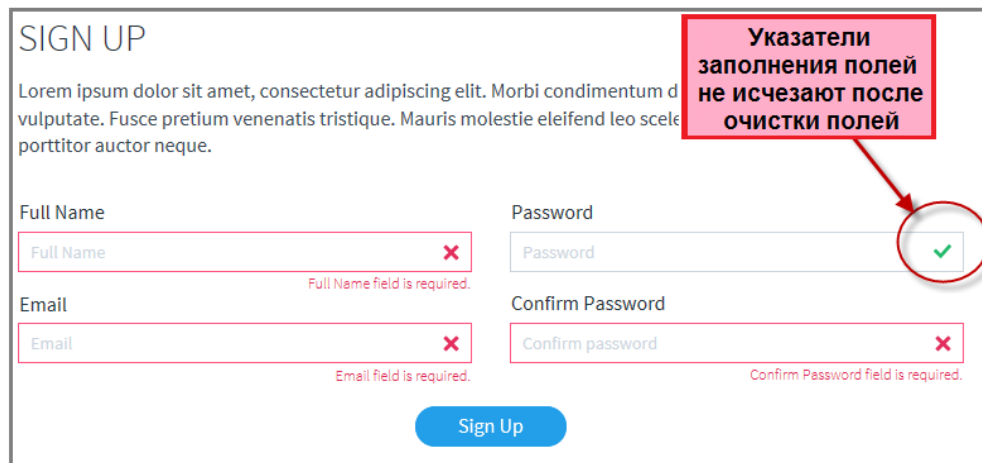


Рисунок 5.1 – Пример скриншота, поясняющего обнаруженный дефект

Делать снимок всего окна программы необязательно: на снимке должна быть видна ошибка и место, в котором она находится. Если для понимания дефекта необходим контекст, то помимо собственно ошибки фиксируют необходимую информацию (браузер, в котором открыто web-приложение, все окно программы в фоне с названием диалогового окна, где эта ошибка появилась). Если необходимо привести снимки нескольких страниц проекта, связанных между собой, лучше сделать это на одном скриншоте, совместив изображения по горизонтали и при необходимости отметив стрелками переходы значений, полей и т. п.

Далее приведены рекомендации по описанию дефектов.

Часто сообщение об ошибке превращается в сокращенную запись только основных действий, необходимых для воспроизведения ошибки, опуская все несущественные. Но, будучи незнакомым с внутренней структурой приложения, тестировщик не может знать, какие из выполненных им действий наиболее существенны для диагностирования данной ошибки. Пренебрегая действиями, которые кажутся незначительными, повышается риск потери важной информации. Лучший способ избежать этой проблемы состоит в том, чтобы просто перечислить все действия, которые необходимы для воспроизведения ошибочного поведения, начиная с открытия нужной формы в проекте.

Если есть подозрение на повторение дефекта в нескольких модулях проекта, этот факт нужно исследовать еще до внесения дефекта и при его описании указать все места, где дефект воспроизводится.

Дефект не должен содержать фразу: «Это не работает», дефект должен показать, что и при каких условиях не работает.

Чтобы внести дефект, его следует воспроизвести минимум два раза, причем начиная с самых нейтральных условий воспроизведения, и только после гарантированного повторения описать последовательность действий.

Нельзя не описывать дефекты только потому, что их не получается воспроизвести. Факт невозможности выяснить причину дефекта в таком случае обязательно должен быть указан в описании дефекта.

Дефекты целесообразно группировать, однако делать это необходимо в соответствии с приведенными ниже правилами.

GUI дефекты могут группироваться в один по признаку формы, на которой они находятся, т. е. если одна форма содержит несколько GUI дефектов с одинаковым уровнем Severity, то их можно объединить.

Функциональные дефекты группируются в том случае, если речь идет об однотипных дефектах, которые воспроизводятся в различных модулях, страницах или полях (например, динамическое обновление не работает в модулях 1, 2 и 4 или отсутствует валидация на спецсимволы на всех полях страницы).

Группировка функциональных дефектов по признаку формы, на которой они найдены, не применяется.

Недопустимо объединять в один дефекты разного типа, например функциональные и GUI. В таком случае пишутся несколько дефектов на каждый тип.

Рекомендации по хорошему описанию дефектов:

1. Шаги воспроизведения, фактический и ожидаемый результаты должны быть подробно описаны.

2. Дефект должен быть понятно описан (с использованием общеупотребимой лексики, точных названий программных средств).

3. Необходимо давать ссылку на соответствующее требование, к нарушению которого приводит фактический результат работы программного средства.

4. Если существует какая-либо информация, которая поможет быстрее обнаружить или исправить дефект, необходимо сообщить эту информацию.

5. Окружение (ОС, браузер, настройки и т. п.), под которым возникла ошибка, должно быть четко указано.

6. Создавать дефект и описывать его необходимо сразу же, как только он был обнаружен. Откладывание «на потом» приводит к риску забыть о дефекте или каких-либо деталях его воспроизведения. Несвоевременное создание дефекта не позволяет проектной команде реагировать на ее обнаружение в реальном времени.

7. После описания дефекта необходимо еще раз перечитать его, убедиться, что все необходимые поля заполнены и все написано верно.

Помимо собственно описания дефектов, результаты тестирования вносят в рабочую тестовую документацию (Acceptance Sheet, Test Survey, Test Cases). Для этого напротив выполненной проверки указывают степень критичности обнаруженного дефекта, его номер и заголовок. Если по результатам конкретной проверки выявлено несколько дефектов, то перечисляют номера всех дефектов, а в качестве степени критичности и заголовка указывают наиболее серьезный дефект.

Практическое задание:

1. Выбрать объект реального мира (например: холодильник, блендер, лифт и др.), выделить в нем модули.

2. Разработать 20 и более тестовых проверок для выбранного объекта реального мира с указанием тестируемого модуля и глубины тестового покрытия (Smoke, MAT, AT).

3. Сформулировать по два возможных дефекта на каждый уровень Severity (Critical, Major, Average, Minor, Enhancement) для выбранного объекта реального мира.

4. Описать по одному дефекту на каждый уровень Severity (Critical, Major, Average, Minor, Enhancement) для выбранного объекта реального мира.

5. Протестировать web-приложение в соответствии с составленной ранее тестовой документацией.

6. Описать все найденные дефекты в отчете о дефектах в среде Microsoft Excel.

7. В отчете о дефектах указать номер тестируемой сборки, название приложения, период времени тестирования, ФИО тестировщика, тестовое окружение (операционная система, браузер).

8. Для каждого дефекта указать его порядковый номер, заголовок, важность, алгоритм воспроизведения, фактический результат, ожидаемый результат, приложение, примечание.

9. Для каждого дефекта обязательно сделать скриншоты.

10. В рабочую тестовую документацию внести результаты тестирования с указанием напротив соответствующей проверки степени критичности обнаруженного дефекта, его номера и заголовка.

11. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.

2. Отчет о результатах тестирования выбранного объекта реального мира с перечислением тестовых проверок, сформулированных дефектов на каждый уровень Severity, описания дефектов.

3. Отчет о найденных дефектах web-приложения.

4. Рабочая тестовая документация с внесенными дефектами web-приложения.

5. Выводы по работе.

Контрольные вопросы:

1. Что такое дефект?

2. Какие характеристики необходимо указать при описании дефекта?

3. Что такое *Headline/Summary* в описании дефекта?

4. На какие три вопроса должен отвечать *Headline/Summary*?

5. Что такое *Severity* в описании дефекта?

6. Какие существуют степени *Severity*? Приведите примеры.

7. Что такое *Description* в описании дефекта?

8. Что такое *Expected result* в описании дефекта?

9. Зачем нужен *Attachment* при описании дефекта?

10. Какие существуют рекомендации по описанию дефектов?

11. Какие дефекты можно группировать?

Лабораторная работа №6

Документирование результатов тестирования

Цель: составить итоговый отчет о результатах тестирования web-приложения.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Итоговый отчет о качестве проверенного функционала является неотъемлемой частью работы, которую каждый тестировщик должен выполнить по завершению тестирования.

Итоговый отчет можно разделить на части с соответствующей информацией:

1. Общая информация.
2. Сведения о том, кто и когда тестировал программный продукт.
3. Тестовое окружение.
4. Общая оценка качества приложения.
5. Обоснование выставленного качества.
6. Графическое представление результатов тестирования.
7. Детализированный анализ качества по модулям.
8. Список самых критичных дефектов.
9. Рекомендации.

Пример итогового отчета приведен на рисунке 6.1.

Далее рассмотрим подробно каждую часть итогового отчета.

Общая информация включает:

- название проекта;
- номер сборки;
- модули, которые подверглись тестированию (в случае, если тестировался не весь проект);
- виды тестов по глубине покрытия (Smoke Test, Minimal Acceptance Test, Acceptance Test), тестовые активности (New Feature Test, Regression Testing, Defect Validation);
- количество обнаруженных дефектов;
- вид рабочей тестовой документации (Acceptance Sheet, Test Survey, Test Cases).

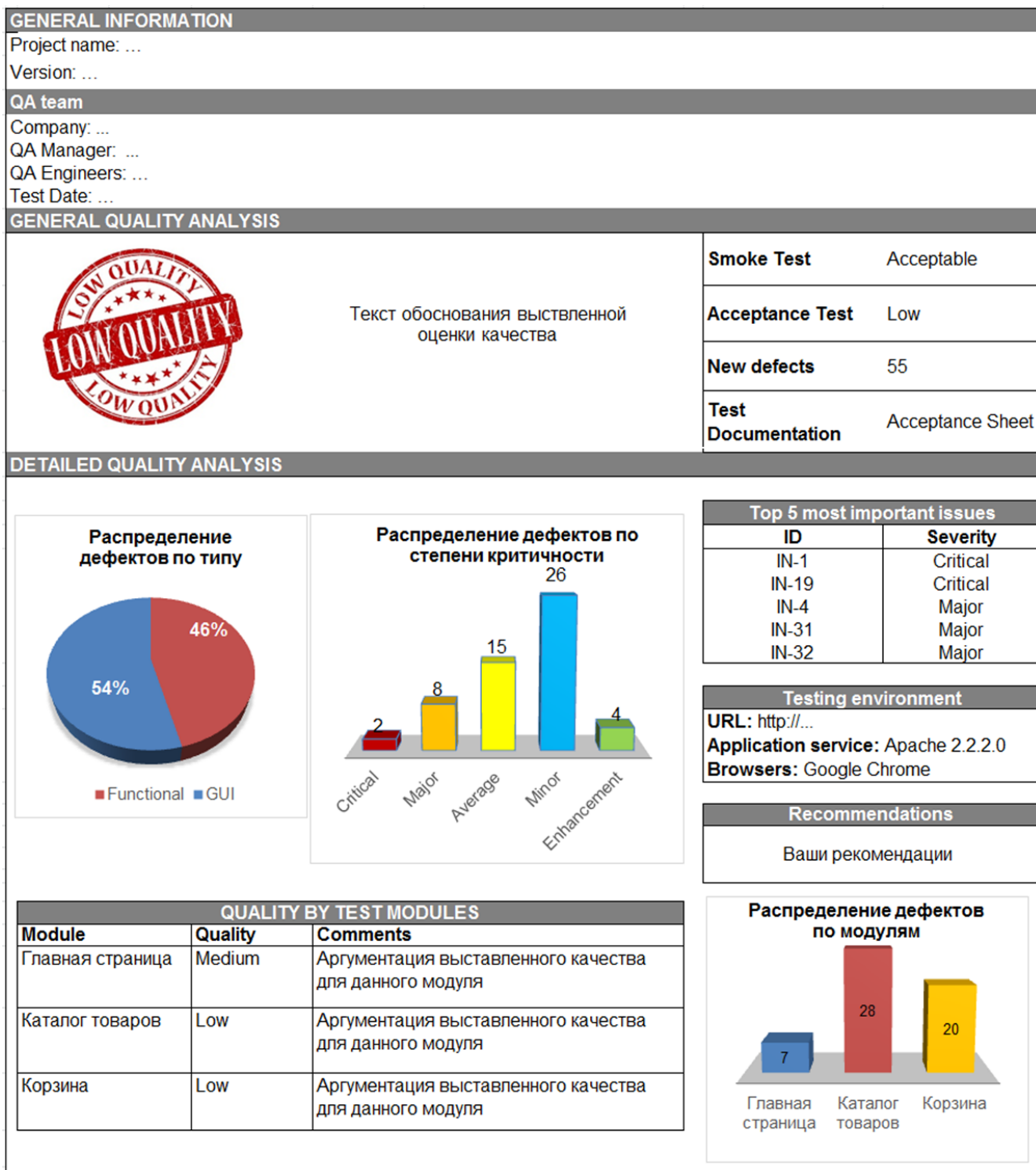


Рисунок 6.1 – Пример итогового отчета о результатах тестирования

Сведения о том, кто и когда тестировал программный продукт, включают информацию о команде тестирования с указанием контактных данных и временном интервале тестирования.

Тестовое окружение содержит ссылку на проект, браузер, операционную систему и другую информацию, конкретизирующую особенности конфигурации.

Общая оценка качества приложения выставляется на основании анализа результатов работы с приложением, количества внесенных дефектов, важности дефектов. Обязательно учитывается этап разработки проекта – то, что не критично в начале работы, становится важным при выпуске программного продукта. Уровни качества: высокое (High), среднее (Medium), низкое (Low).

Обоснование выставленного качества является наиболее важной частью отчета, т. к. здесь отражается общее состояние сборки, а именно:

- качество сборки на текущий момент;
- факторы, повлиявшие на выставление именно такого качества сборки: указание функционала, который заблокирован для проверки, перечисление наиболее критичных дефектов и объяснение их важности для пользователя или бизнеса заказчика;
- анализ качества проверенного функционала: улучшилось оно или ухудшилось по сравнению с предыдущей версией;
- если качество сборки ухудшилось, то обязательно должны быть указаны регрессионные места;
- наиболее нестабильные части функционала с указанием причин, по которым они таковыми являются.

Помимо вышеуказанных общих характеристик при выставлении и обосновании оценки качества программного продукта активно используются числовые характеристики качества – метрики. Пример обоснования с использованием метрик выглядит следующим образом: «Реализовано 79 % требований (в том числе 94 % важных), за последние три билда тестовое покрытие выросло с 63 до 71 %, а общий показатель прохождения тест-кейсов вырос с 85 до 89 %».

Метрики могут быть как прямыми (не требуют вычислений), так и расчетными (вычисляются по формуле). Типичные примеры прямых метрик – количество разработанных тест-кейсов, количество найденных дефектов и т. д.

Большинство общепринятых расчетных метрик могут быть собраны автоматически с использованием инструментальных средств управления проектами:

- процентное отношение (не)выполненных тест-кейсов ко всем имеющимся;
- процентный показатель успешного прохождения тест-кейсов;
- процентный показатель заблокированных тест-кейсов;
- плотность распределения дефектов;
- эффективность устранения дефектов;

- распределение дефектов по важности и срочности;
- метрики покрытия.

Простая расчетная метрика для определения показателя успешного прохождения тест-кейсов выглядит следующим образом:

$$T^{SP} = \frac{T^{Success}}{T^{Total}} \cdot 100 \%, \quad (6.1)$$

где $T^{Success}$ – количество успешно выполненных тест-кейсов;
 T^{Total} – общее количество выполненных тест-кейсов.

Минимальные границы значений показателя успешного прохождения тест-кейсов на начальной фазе проекта равны 10 %, на основной фазе проекта – 40 %, на финальной фазе проекта – 85 %.

Метрику покрытия требований тест-кейсами вычисляют по формуле

$$R^{SimpleCoverage} = \frac{R^{Covered}}{R^{Total}} \cdot 100 \%, \quad (6.2)$$

где $R^{Covered}$ – количество требований, покрытых хотя бы одним тест-кейсом;
 R^{Total} – общее количество требований.

Метрики являются мощнейшим средством сбора и анализа информации. Однако использование метрик требует ясного понимания их сущности, в противном случае может возникнуть ситуация использования «метрик ради метрик», когда многочисленные цифры и графики никто не может понять и правильно интерпретировать.

Графическое представление результатов тестирования способствует более полному и быстрому пониманию текстовой информации (см. рисунок 4.1).

Если необходимо продемонстрировать процентное соотношение, то целесообразно использовать круговые диаграммы (например, процентное соотношение функциональных дефектов и дефектов GUI).

Столбчатые диаграммы лучше подойдут там, где важно визуализировать количество дефектов в зависимости от степени их критичности или в зависимости от локализации (распределение дефектов по модулям).

Отразить в итоговом отчете динамику качества по всем сборкам лучше всего удастся с помощью линейного графика.

Детализированный анализ качества по модулям

В данной части отчета описывается более подробная информация о проверенных частях функционала, устанавливается качество каждой проверенной части функционала (модуля) в отдельности, дается аргументация выставленного уровня качества. Как правило, данный раздел отчета представляется в табличной форме. В зависимости от вида проводимых тестовых активностей эта часть отчета будет отличаться.

При оценке качества функционала на уровне Smoke теста оно может быть либо приемлемым (Acceptable), либо неприемлемым (Unacceptable). Если все наибо-

лее важные функции работают корректно, то качество всего функционала на уровне Smoke может быть оценено, как приемлемое.

Если это релизная или пререлизная сборка, то для выставления приемлемого качества на уровне Smoke не должно быть найдено функциональных дефектов.

В части о детализированной информации качества сборки следует более подробно описать проблемы, которые были найдены во время теста.

При оценке качества функционала на уровне Defect Validation указываются результаты валидации дефектов, а именно:

- общее количество всех дефектов, поступивших на проверку;
- количество неисправленных дефектов и их процент от общего количества;
- список дефектов, которые не были проверены и причины, по которым этого не было сделано;
- наглядная таблица с неисправленными дефектами.

По вышеуказанным результатам выставляется качество теста. Если процент неисправленных дефектов меньше 10 %, то качество приемлемое (Acceptable), если больше 10 %, то качество неприемлемое (Unacceptable).

При оценке качества функционала на уровне New Feature Test (полный тест нового функционала) качество отдельного проверенного функционала может быть высоким (High), среднее (Medium), низкое (Low).

Важно отдельно указывать информацию о качестве каждого модуля нового функционала с аргументацией выставленной оценки.

При оценке качества функционала на уровне Regression Testing нужно анализировать динамику изменения качества проверенной функциональности в сравнении с более ранними версиями сборки. Для этого приводится сравнительная характеристика каждой из частей функционала в сравнении с предыдущими версиями сборки, даются ясные пояснения о выставлении соответствующего качества каждой функции в отдельности. Так же как и у предыдущего вида тестов, качество этих может быть высоким (High), среднее (Medium), низкое (Low).

Список самых критичных дефектов содержит 3–5 ссылок на наиболее критичные дефекты с указанием их названия и уровня критичности.

Рекомендации включают краткую информацию о всех проблемах приложения с пояснениями, насколько оставшиеся проблемы являются критичными для конечного пользователя. Обязательно указывают функционал и дефекты, скорейшее исправление которых является наиболее приоритетным. Кроме того, если сборка является релизной или пререлизной, то любое ухудшение качества является критичным и важно это обозначить.

Практическое задание:

1. Составить итоговый отчет по результатам тестирования web-приложения.

2. Указать общую информацию о тестируемом продукте (название, номер сборки, виды выполненных тестов, количество обнаруженных дефектов, вид рабочей тестовой документации).
3. Указать, кто и когда тестировал программный продукт.
4. Описать тестовое окружение (ссылку на web-приложение, браузер).
5. Указать общую оценку качества протестированного приложения и подробно ее обосновать.
6. Графически (в виде круговой диаграммы) отразить процентное соотношение дефектов GUI и функциональных дефектов.
7. Графически (в виде столбчатой диаграммы) отразить распределение дефектов по различным степеням критичности.
8. Графически (в виде столбчатой диаграммы) отразить распределение дефектов по модулям.
9. Произвести детальный анализ качества всех модулей протестированного приложения с аргументацией выставленных уровней качества.
10. Привести список пяти наиболее критичных дефектов.
11. Сформулировать рекомендации по улучшению качества программного продукта.
12. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Итоговый отчет о результатах тестирования web-приложения.
3. Выводы по работе.

Контрольные вопросы:

1. Какова структура итогового отчета о результатах тестирования?
2. Что содержится в разделе Общая информация?
3. Что содержится в разделе Тестовое окружение?
4. Как выставляется общая оценка качества приложения?
5. Как обосновать выставленную оценку качества?
6. Что такое метрика в тестировании?
7. Приведите примеры прямых метрик.
8. Приведите примеры расчетных метрик.
9. Для чего используется графическое представление результатов тестирования в итоговом отчете?
10. Что содержится в разделе Детализированный анализ качества?
11. Что содержится в разделе Рекомендации?

Лабораторная работа №7

Тестирование юзабилити

Цель: изучить и реализовать на практике экспертный и пользовательский подходы юзабилити-тестирования.

План занятия:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Юзабилити – степень, с которой продукт может быть использован определенными пользователями при определенном контексте для достижения определенных целей с должной эффективностью, результативностью и удовлетворенностью. Юзабилити отражает степень удобства использования программного продукта конечными пользователями. Так как взаимодействие пользователя и программного обеспечения осуществляется посредством пользовательских интерфейсов, то понятие юзабилити прежде всего относится к процессу разработки пользовательских интерфейсов.

Юзабилити-тестирование позволяет сделать программный продукт более простым и удобным в использовании, тем самым не только повышая эффективность работы конечных пользователей и бизнес-процессов в целом, но и улучшая впечатление от взаимодействия с программным обеспечением.

Для выявления проблем удобства использования, в том числе на ранних этапах планирования и разработки программных продуктов, используются два основных подхода:

1. Проверка соответствия принципам обеспечения удобства пользования и корректного визуального представления в контексте функциональных требований посредством экспертной оценки (экспертный подход).
2. Изучение опыта взаимодействия пользователя с приложением через имитацию поведения пользователей (пользовательский подход).

Для юзабилити-тестирования одного программного обеспечения могут применяться оба подхода (методика двойной проверки).

Далее рассмотрим вышеуказанные техники юзабилити-тестирования более подробно.

Экспертный подход юзабилити-тестирования

При экспертном подходе в качестве пользователей выступают два и более экспертов (оптимальное количество для больших проектов 5–6 человек). Эксперты проходят основные сценарии поведения пользователей и анализируют их с точки зрения:

- стандартов юзабилити для конкретного типа программного продукта (например, Android Material Design для мобильных приложений на платформе Android);
- общих принципов юзабилити (эвристики Якоба Нильсена);
- здравого смысла и опыта.

По результатам прохождения пользовательских сценариев составляется отчет о дефектах.

Преимущества экспертного подхода:

- быстрый в применении;
- эксперты гарантировано понимают общие задачи программного продукта.

Недостаток данного подхода – субъективизм (эксперты не являются реальными пользователями).

Ниже приведены основные принципы юзабилити, сформулированные Якобом Нильсеном:

1. Информированность о состоянии системы. Пользователь всегда должен ориентироваться и четко понимать, что происходит в системе. Взаимодействие между пользователем и системой должно быть как можно более логичным и быстрым. Для этого целесообразно реализовать обратную связь в виде сообщений подтверждения успешности выполнения действий, запросов на подтверждение удаления, сообщений об ошибках и др.

2. Схожесть системы с реальным миром. Система должна общаться с пользователем на понятном ему языке. Использование инфографики, слов, фраз и понятий, знакомых пользователю в реальном мире, намного предпочтительнее, чем использование специализированных терминов.

3. Свобода действий. Необходимо предоставить пользователям возможность отмены действий, а также возврата к ранее осуществленным действиям.

4. Единообразие и стандарты. Не следует вводить в заблуждение пользователя, описывая одни и те же вещи разными словами и терминами.

5. Предотвращение ошибок. Важно свести к минимуму количество условий, в которых могут быть допущены ошибки. Например, можно давать пользователям подсказки, поясняющие, какую информацию надо вводить в текстовые поля.

6. На виду, а не в памяти. Не следует вынуждать пользователя запоминать большое количество объектов, действий и опций. Вся необходимая информация должна быть размещена в пределах доступности для пользователя.

7. Гибкость и эффективность. Не следует нагружать пользователей лишней информацией – предоставьте им возможность совершать часто повторяющиеся действия как можно быстрее и проще.

8. Эстетичный и минималистичный дизайн. Тексты не должны содержать бесполезной или устаревшей информации. Каждое лишнее слово делает восприятие все более трудным и лишает посетителя возможности достичь цели.

9. Понимание проблем и их решение. Сообщения об ошибках должны быть выражены на понятном пользователю языке, как можно более точно описывать проблему и предоставлять возможные варианты ее решения.

10. Справочные материалы и документация. Даже если система может использоваться без документации, в процессе работы с ней все же может потребоваться справочная информация. Подобные документы должны составляться таким образом, чтобы в них легко было найти необходимое.

Пользовательский подход

Пользователям (3–5 человек из каждого сегмента целевой аудитории), согласившимся участвовать в тестировании, предлагают пройти наиболее распространенные и наиболее проблемные сценарии. Эксперт протоколирует действия пользователя, фиксирует все в видеоформате, чтобы отследить реакцию (эмоции) пользователя, но никак не влияет на действия пользователя.

Преимущества пользовательского подхода:

- объективные результаты (участвуют реальные пользователи);
- процесс легко измерим.

Возможные измерения при юзабилити-тестировании:

- время выполнения задачи;
- успешность выполнения задачи;
- эффект первого впечатления (например, сколько раз улыбнулся).

Недостатки пользовательского подхода:

- длительный по времени;
- дорогой (если пользователей привлекают на платной основе);
- большое внимание следует уделить подбору пользователей.

Для реализации пользовательского подхода юзабилити-тестирования необходимо провести предварительную работу, которая включает следующие этапы:

1. Определение цели пользователя, цели бизнеса.
 2. Исследование целевой аудитории: составление ее общего портрета, сегментация на группы, описание персонажей как ярких представителей каждой группы.
 3. Выявление контекста – ситуаций, при которых пользователь обращается к программному продукту.
 4. Составление пользовательских сценариев.
- Рассмотрим особенности перечисленных этапов.

Цели пользователя и цели бизнеса выясняют посредством общения с заказчиком и потенциальными потребителями, анализа спецификации, исследования существующих аналогов.

Исследование целевой аудитории необходимо для проектирования взаимодействия в целом и корректировки элементов интерфейса.

Целевая аудитория – группа пользователей, на которую ориентировано содержание программного продукта. При исследовании целевой аудитории на первом этапе необходимо составить ее общий портрет, указав следующие характеристики:

1. Социально-демографические характеристики: пол, возраст, образование, уровень дохода, сфера деятельности, семейное положение.

2. Психологические характеристики: стиль жизни, особенности личности, черты характера, жизненная позиция, система ценностей. Более ценная информация для проектирования, чем первая группа критериев. Например, если известно, что целевая аудитория больше всего ценит время, можно спроектировать простой интерфейс и дать возможность получать не весь контент, а самое ценное для конкретной целевой группы, или предоставить инструменты персонализации каждого человека.

3. Поведенческие характеристики: повод для регистрации, искомые выгоды, частота посещаемости конкурентов, степень готовности к переходу на другой продукт, отношение к проекту (если он не новый) и т. д. Поведенческие характеристики целевой аудитории помогают понять привычки, мотивацию, круг интересов, проблемы, надежды и ожидания пользователя (как именно человек делает выбор, что влияет на решение купить продукт или отказаться от покупки, какой параметр является главным: качество товара, известная марка, стоимость, мнение друзей). Собрать эти данные очень сложно. Эта информация может быть у заказчика или конкурента, если проектируется новая версия уже существующего проекта, либо ее нужно будет собирать через опросы целевой аудитории или составление карт эмпатии.

4. Географические характеристики: страна, город, район. Если стоит задача по проектированию национальных продуктов или продуктов с геолокацией, то важность этих характеристик резко возрастает.

Для получения данных, составляющих общий портрет целевой аудитории, используют различные способы сбора информации:

1. Статистика запросов поисковых систем: позволяет оценить величину целевой аудитории по числу поисковых запросов.

2. Анализ данных из социальных сетей и других публичных источников.

3. Опросы аудитории (анкетирование пользователей, использование опросной формы или регистрации на уже функционирующем сайте).

4. Данные счетчика посещений: для уже функционирующих web-ресурсов позволяют изучить все действия пользователей и конкретизировать распределение аудитории по регионам, времени и др.

5. Составление карты эмпатии (рисунок 7.1).



Рисунок 7.1 – Карта эмпатии

Эмпатия – это психологический термин, который отражает способность понимать чувства и настроения других людей, умение поставить себя на место другого.

Карта эмпатии – это метод исследования целевой аудитории, направленный на составление подробного портрета типичного пользователя для конкретного программного продукта. Карта эмпатии представляет собой диаграмму профиля пользователя. Блоки «что видит» и «что слышит покупатель» выявляют оптимальные каналы распространения информации о разрабатываемой системе. Особое внимание следует уделять последним двум блокам карты эмпатии: итоговый программный продукт должен развеять все сомнения и тревоги, заключенные в блоке «боль», и максимально подчеркивать способность помочь в реализации це-

лей из блока «достижения». Важно уделить внимание возможному конфликту между тем, что человек «говорит и делает» на публике, и тем, как он «думает и чувствует» на самом деле.

После общего описания целевой аудитории выполняется ее детализация (если она неоднородна) посредством сегментации на группы с общими параметрами.

Визуализация каждой группы производится с помощью составления конкретных «персонажей» (персон) как типичных представителей различных групп целевой аудитории. Персонаж — это реалистичный собирательный образ пользователя, представляющий один сегмент целевой аудитории. Портрет персонажа включает: фотографию, имя, возраст, пол, образование, профессию, семейное положение, личностные характеристики, которые могут повлиять на взаимодействие с программным обеспечением, взгляды и интересы. При взаимодействии персонажа с информационной системой он преследует конкретные цели, которые необходимо выявить и отразить в портрете персонажа. Описание особенностей взаимодействия персонажа с программным продуктом целесообразно построить в виде ответов на вопросы: в какой обстановке персонаж будет использовать разрабатываемый продукт; как часто он будет обращаться к данному продукту; был ли опыт использования подобных информационных систем; каковы ожидаемые результаты от взаимодействия? После описания персонажей необходимо проверить их на полноту и избыточность описания.

Среди сформированных персонажей важно отличать ключевых (именно для них будет происходить проектирование информационной системы) и второстепенных (их потребности необходимо учитывать, но реализация этих потребностей не должна мешать ключевым персонажам реализовывать свои цели).


Завершающим подготовительным этапом пользовательского подхода юзабилити-тестирования является разработка карты сценариев. Сценарий поведения — это ситуация взаимодействия персонажа с продуктом. Описание сценария включает название, шаги, входную/выходную информацию, пожелания/предложения, эмоции, комментарии. Сценарий нужно описать полностью (от начала до логического завершения) и максимально правдоподобно.

После выполнения вышеописанных этапов для каждого сегмента целевой аудитории подбирают 3–5 представителей в соответствии с составленными персонажами и приступают собственно к тестированию: пользователи проходят распространенные и наиболее проблемные сценарии под наблюдением эксперта, который фиксирует время и успешность выполнения сценария, эмоциональные впечатления. По результатам тестирования представляется отчет.

Практическое задание:

1. Выбрать программное обеспечение для проведения юзабилити-тестирования.
2. Провести юзабилити-тестирование на основе экспертного подхода, руководствуясь эвристиками Якоба Нильсена.
3. По результатам юзабилити-тестирования на основе экспертного подхода составить отчет о дефектах.
4. Провести юзабилити-тестирование на основе пользовательского подхода.
5. В рамках пользовательского подхода определить цель заказчика, цель пользователя.
6. Составить общую характеристику целевой аудитории.
7. Разработать карту эмпатии для типичного представителя целевой аудитории в соответствии с рисунком 7.1.
8. Сегментировать целевую аудиторию в зависимости от параметров, наиболее влияющих на исследуемую информационную систему (пол, возраст, род занятий и др.).
9. Для каждого сегмента целевой аудитории в соответствии с таблицей 7.1 составить портрет персонажа – наиболее типичного представителя данной группы.

Таблица 7.1 – Профиль персонажа

<Категория целевой аудитории>	
<Роль персоны>	
 Фотография	Описание <В описании указываются следующие характеристики: ФИО, пол, возраст, род занятий, семейное положение, образование, увлечения, социальный статус, место работы>
Личные характеристики	
Цели: <Перечень целей, которые пользователь стремится достичь во время использования системы>	
Взаимодействие с продуктом <Рабочий процесс и контекст (окружение)>	
Неудовлетворенности и ожидания <Описание исключительных ситуаций и вытекающих из них проблем, дополнительных возможностей программного продукта>	

10. Разработать типичные сценарии взаимодействия ключевого персонажа с программным продуктом в соответствии с таблицей 7.2.

Таблица 7.2 – Описание сценария

Шаг	<название шага 1>
Вопросы	<перечислить все вопросы, которые могут возникнуть у персонажа при работе с продуктом на данном шаге>
Пожелания	<перечислить пожелания, которые могут возникнуть у персонажа при работе с продуктом на данном шаге >
Эмоции	<какие эмоции возникнут у персонажа от взаимодействия с продуктом>
Комментарии	<описание окружения, исходных данных>

11. Отобрать по одному представителю от каждой выделенной группы целевой аудитории в соответствии с составленными персонажами.

12. Пронаблюдать за тем, как потенциальные пользователи выполняют типичные сценарии взаимодействия с программным обеспечением.

13. По результатам наблюдений составить отчет с указанием времени и успешности выполнения сценариев, эмоциональных впечатлений пользователей.

14. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Отчет о дефектах по результатам юзабилити-тестирования на основе экспертного подхода.
3. Результаты подготовки к юзабилити-тестированию на основе пользовательского подхода: общее описание целевой аудитории, карта эмпатии типичного представителя целевой аудитории, сегментирование целевой аудитории, портреты персонажей каждого сегмента, типичные сценарии взаимодействия пользователей с программным обеспечением.
4. Отчет о результатах юзабилити-тестирования на основе пользовательского подхода.
5. Выводы по работе.

Контрольные вопросы:

1. Что такое юзабилити?
2. Какие существуют подходы к юзабилити-тестированию?
3. Охарактеризуйте экспертный подход юзабилити-тестирования.

4. Перечислите и дайте характеристику принципам юзабилити Якоба Нильсена.
5. Охарактеризуйте пользовательский подход юзабилити-тестирования.
6. Что такое целевая аудитория?
7. Для чего необходимо изучать целевую аудиторию?
8. Какие существуют способы сбора информации о целевой аудитории?
9. Что такое эмпатия? Что такое карта эмпатии?
10. Из каких характеристик состоит профиль персонажа?
11. Какой персонаж называют ключевым, а какой второстепенным?
12. Что такое сценарий?

Литература

1. Справочные материалы компании AIQA [Электронный ресурс]. – 2016. – 1 электрон. опт. диск (CD-ROM).
2. Куликов, С. С. Тестирование программного обеспечения. Базовый курс : практ. пособие / С. С. Куликов. – Минск : Четыре четверти, 2015. – 294 с.
3. Савин, Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. – М. : Дело, 2007. – 312 с.
4. ISTQB. Стандартный глоссарий терминов, используемых в тестировании программного обеспечения. – 2014. – 73 с.
5. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс. 3-е изд., доп.; пер. с англ. – М. : Русская редакция ; СПб. : БХВ-Петербург, 2014. — 736 с.
6. Блэк, Р. Ключевые процессы тестирования / Р. Блэк. – М. : BHV, 2008.
7. Тамрэ, Л. Введение в тестирование программного обеспечения / Л. Тамрэ. – М. : Вильямс, 2005.
8. Бек, К. Экстремальное программирование: разработка через тестирование / К. Бек. – СПб. : Питер, 2005.
9. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования ПО / Б. Бейзер. – СПб. : Питер, 2006.
10. Стотлемайер, Д. Тестирование web-приложений / Д. Стотлемайер. – М. : Кудиц-образ, 2003.
11. Copeland, L. A Practitioner's Guide to Software Test Design / L. Copeland. – London : Artech House Publishers, 2004. – 274 p.

Учебное издание

**Меженная Марина Михайловна
Гордейчук Татьяна Валерьевна
Борисик Марина Михайловна и др.**

ЮЗАБИЛИТИ-ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ПОСОБИЕ