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

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

ТЕХНОЛОГИИ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальностей 6-05-0612-01 «Программная инженерия», 6-05-0611-01 «Информационные системы и технологии»

Под редакцией Т. В. Казак

Минск БГУИР 2025

УДК 004.415.5(076)
ББК 32.972.1я73
Т38

А в т о р ы:
А. Н. Василькова, А. В. Воробей, О. С. Медведев, А. М. Прудник

Р е ц е н з е н т ы:
кафедра интеллектуальных информационных технологий
Белорусского государственного университета информатики
и радиоэлектроники
(протокол № 8 от 21.10.2024);

кафедра управления охраной труда Белорусского государственного аграрного
технического университета
(протокол № 5 от 09.12.2024);

заместитель генерального директора по научной работе
государственного научного учреждения
«Объединенный институт проблем информатики
Национальной академии наук Беларуси»
кандидат технических наук, доцент
С. Н. Касанин

Технологии оценки качества программного обеспечения : пособие /
Т38 А. Н. Василькова, А. В. Воробей, О. С. Медведев, А. М. Прудник ; под ред.
Т. В. Казак. – Минск : БГУИР, 2025. – 86 с. : ил.
ISBN 978-985-543-825-1.

Посвящено вопросам анализа, планирования, проведения тестовых испытаний и оценки качества программного обеспечения на всех стадиях его жизненного цикла. Является методическим обеспечением выполнения лабораторных работ для студентов специальностей «Программная инженерия» и «Информационные системы и технологии» всех форм обучения.

УДК 004.415.5(076)
ББК 32.972.1я73

ISBN 978-985-543-825-1

Содержание

Введение

Настоящее пособие содержит лабораторные работы по дисциплине «Технологии оценки качества программного обеспечения», предназначеные для студентов всех форм обучения специальностей «Программная инженерия» («Инженерно-психологическое обеспечение информационных технологий»), «Информационные системы и технологии», а также для студентов других специальностей, направленных на подготовку инженеров-программистов со знанием основ обеспечения качества программных продуктов.

Цель данного пособия заключается в оказании методической помощи студентам в изучении и выполнении лабораторных работ по дисциплине «Технологии оценки качества программного обеспечения» с учетом специфики специальностей профилирующей кафедры инженерной психологии и эргономики.

Пособие посвящено вопросам анализа, планирования, проведения тестовых испытаний и оценки качества программного обеспечения на всех стадиях его жизненного цикла. Подробно раскрывается тема классификации видов тестирования. Изучаются основы планирования тестирования, разработки рабочей тестовой документации, поиска и описания дефектов, оценки качества и документирования результатов тестирования. Отдельное внимание уделено автоматизированному тестированию программного обеспечения: написание автотестов с использованием библиотеки Selenium WebDriver и инструмента Postman (тестирование API).

Цель преподавания учебной дисциплины: изучение базовых принципов, овладение современными технологиями и программными средствами тестирования и оценки качества веб-, desktop- и мобильных приложений, включающими этапы планирования тестирования; тестирования требований; разработки тестовой документации; поиска и описания дефектов; оценки качества программного обеспечения и документирования результатов тестирования.

По окончании изучения данной дисциплины студент должен уметь: разрабатывать план тестирования; проводить анализ, обнаруживать дефекты технической документации; разрабатывать тестовую документацию; выполнять тестовые сценарии, обнаруживать и документировать дефекты, контролировать исправление обнаруженных дефектов; оценивать качество тестируемого программного обеспечения, документировать результаты тестирования; использовать программные средства для тестирования безопасности, производительности, юзабилити, автоматизированного тестирования.

Студент должен владеть: навыками анализа, планирования, проведения тестовых испытаний и оценки качества веб-, desktop- и мобильных приложений.

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 результатами.

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).

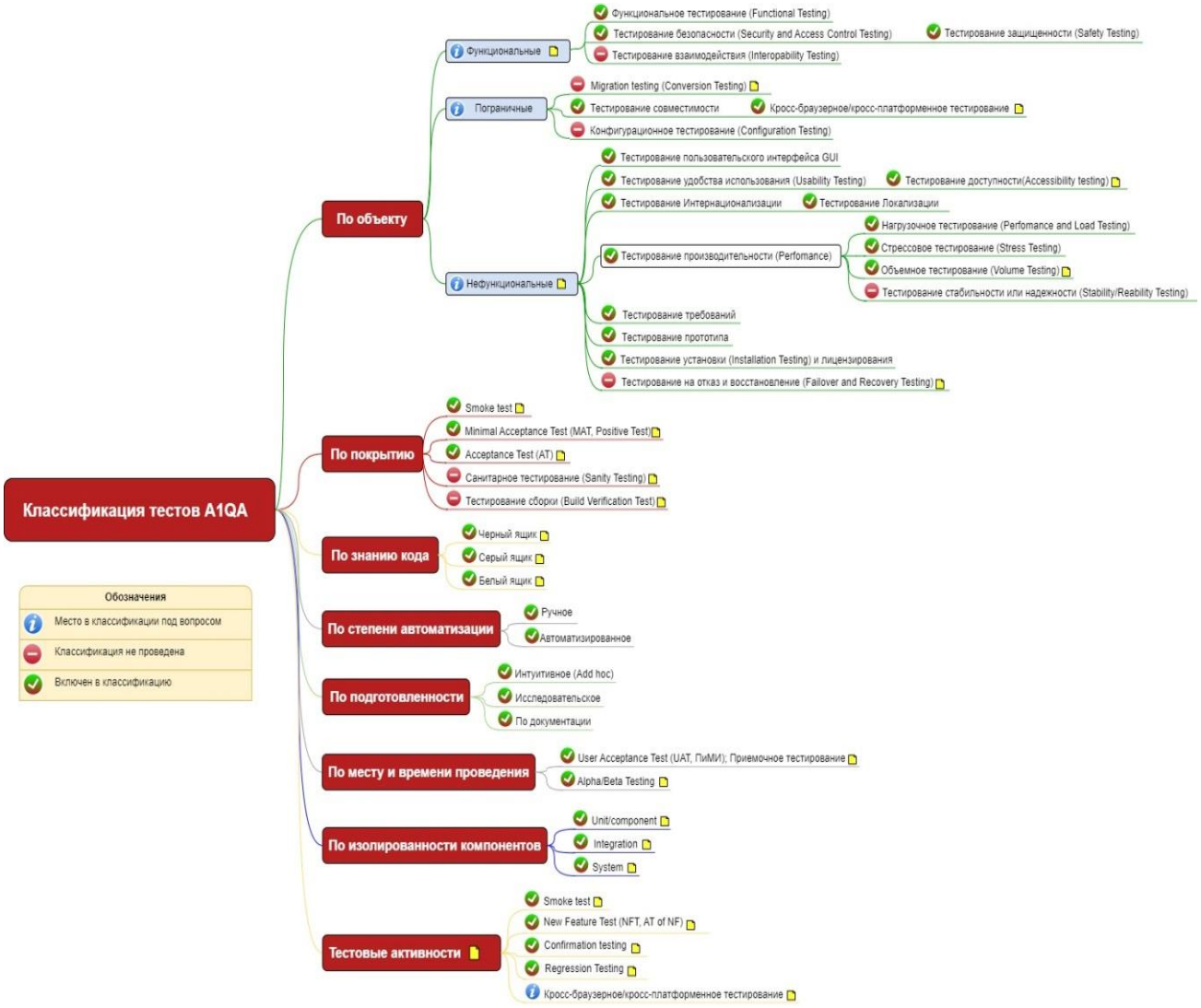

Рисунок 1 – Классификация видов тестирования в зависимости от объекта

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 часто 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: кросс-браузерное тестирование (различные браузеры или версии браузеров), кросс-платформенное 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) [2].

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) – тестирование 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 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.

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, исследовательское, 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 могут 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 собой 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, CT.

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.

Повторное тестирование (Confirmation Testing, CT) – выполнение тест-кейсов, которые ранее обнаружили дефекты, с целью подтверждения устранения дефектов. Фактически этот вид тестирования сводится к действиям на финальной стадии жизненного цикла отчета о дефекте, направленным на то, чтобы перевести дефект в состояние «проверен» и «закрыт».

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 баг-трекинговую систему.

Для последующих поставок программного обеспечения композиции тестов могут быть следующими.

Если не была добавлена новая функциональность, то CT + RT_{MAT}. Иными словами, выполняется проверка исправления дефектов программистом (Confirmation Testing, CT), а также проверка работоспособности остальной функциональности после исправления дефектов на позитивных сценариях (Minimal Acceptance Test, MAT).

Если была добавлена новая функциональность, то Smoke + CT + NFT_{AT} + RT_{MAT}. В частности, выполняется поверхностное тестирование (Smoke Test), проверка исправления дефектов программистом (Confirmation Testing, CT), 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 CT + NFT_{AT} + RT_{MAT}, т. е. без выполнения Smoke Test.

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

Smoke + CT + NFT_{AT} + RT_{MAT}.

В зависимости от типа и специфики приложения (веб, desktop, mobile) выполняют специализированные тесты (например, кросс-браузерное или кросс-платформенное тестирование, тестирование локализации и интернационализации и др.) [5].

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.

Таблица 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

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

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.

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 2).

Рисунок 2 – 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 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 800 × 600…» – 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

способным, эффективно, 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 пользователей (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 кешировать 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 должны быть исключены (или переработаны) 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 [5].

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. Как выглядит жизненный цикл проекта?
2. Какие выделяют критерии качества?
3. Какие требования считаются проверяемыми?
4. Какие требования считаются модифицируемыми?
5. Какие требования считаются корректными?
6. Какие требования считаются недвусмысленными?
7. Какие требования считаются полными?
8. Какие требования считаются непротиворечивыми?
9. Какие требования считаются упорядоченными по важности и стабильности?
10. Какие требования считаются трассируемыми?
11. Какие существуют методы тестирования требований?

Лабораторная работа № 3

Тестирование программного обеспечения: разработка тестов

Цель: разработать рабочую тестовую документацию для тестирования веб-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 2):

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.

При составлении рабочей тестовой документации необходимо указать номер тестируемой сборки, тип выполняемой тестовой активности, период времени тестирования, Ф. И. О. 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 завершенные блоки этого окна.

Таблица 2 – 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

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 3.

Таблица 3 – Примеры Check List, Acceptance Sheet, Test Survey

Вид рабочей тестовой документации	Пример	Глубина тестирования
Check List	Протестировать форму авторизации	–
Acceptance Sheet	Форма авторизации: 1. GUI	AT
	2. General	AT
	3. Поле «Эл. адрес»	AT
	4. Поле «Пароль»	AT
	5. Кнопка «Войти»	AT
	6. Чекбокс «Не выходить из системы»	AT
	7. Ссылка «Забыли пароль»	AT
Test Survey	Форма авторизации: 1. GUI	AT
	2. General	AT
	3. Валидный эл. адрес + валидный пароль	Smoke
	4. Валидный эл. адрес с пробелами в начале и в конце + валидный пароль с пробелами в начале и в конце	MAT
	5. Валидный эл. адрес с пробелами в середине + валидный пароль с пробелами в середине	MAT
	6. Валидный эл. адрес минимально допустимой длины + валидный пароль минимально допустимой длины	MAT
	7. Валидный эл. адрес максимально допустимой длины + валидный пароль максимально допустимой длины	MAT
	8. Пустое поле эл. адреса + пустое поле пароля	AT
	9. Валидный эл. адрес + невалидный пароль (спецсимволы)	AT
	10. Валидный эл. адрес + невалидный пароль (русские буквы)	AT
	11. Валидный эл. адрес + невалидный пароль (длина превышает максимально допустимую)	AT
	12. Валидный эл. адрес + невалидный пароль (длина меньше минимально допустимой)	AT
	13. Невалидный эл. адрес (спецсимволы кроме ’_’, ’-’, ’@’, ’. ’) + валидный пароль	AT
	14. Невалидный эл. адрес (русские буквы) + валидный пароль	AT
	15. Невалидный эл. адрес (использование символа ’@’ дважды) + валидный пароль	AT
	16. Невалидный эл. адрес (отсутствие символа ’.’) + валидный пароль	AT
	17. Невалидный эл. адрес (длина превышает максимально допустимую) + валидный пароль	AT
	18. Невалидный эл. адрес (длина меньше минимально допустимой) + валидный пароль	AT
	19. Запомнить данные: выйти из системы и зайти обратно	MAT
	20. Ссылка «Забыли пароль»	MAT

Тест-кейс (таблица 4) – 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):

- 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 – 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. 3. 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 (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («крайне высокий», «высокий», «средний», «низкий», «крайне низкий») или иным удобным способом. Количество градаций также не фиксировано, но чаще всего лежит в диапазоне от 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. Хоть 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 разработать 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 часто 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 веб-, desktop- и мобильных приложений.

Для любого приложения выполняется тестирование графического интерфейса пользователя (таблица 5).

Таблица 5 – Перечень основных GUI-проверок для всего приложения

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

Общие проверки функциональности для всех типов приложений, а также общие проверки для веб-приложений приведены в таблицах 6, 7.

Таблица 6 – 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»	Элемент навигации, являющийся признаком удобства пользования приложением в целом и перемещением по его структуре
3. Скроллинг	Отсутствие скроллинга в случае, если текст вмещается на странице без прокрутки. Соответствующее изменение текста при использовании скроллинга. Возможность изменения положения скроллинга при помощи мыши, кнопок Page up/down, Home/End
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 7 – Перечень общих проверок функциональности для веб-приложений

Название проверки	Описание проверки
1. Подготовка к тестированию	Перед тестированием каждой новой сборки необходимо осуществить очистку кеша 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. E-mail-нотификации	Проверка работоспособности отправки e-mail-нотификаций (как администратору, так и пользователю), если только отсутствие писем не является спецификой проекта
5. Отображение flash-элементов при отключенном или неустановленном 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 должны работать

В таблицах 8–17 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 8 – Перечень основных проверок для поля ввода данных

Functional Test	GUI Test
1. Обязательность ввода. 2. Обработка только пробелов. 3. Использование пробелов в тексте: 3.1. Пробелы в начале и в конце строки должны отсекаться при сохранении. 3.2. Пробелы внутри текста отсекаться 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, e-mail). 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)	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 9 – Перечень основных проверок для поля загрузки файлов

Functional Test	GUI Test
1. Обязательность выбора файла. 2. Форматы: корректные/некорректные. 3. Корректный формат, но отсутствует/модифицировано расширение. 4. Ограничения на размер (включая загрузку файлов нулевого размера, большого размера). Загрузка исполняемых файлов (EXE, PHP, JSP др.). 5. Загрузка переименованного EXE-файла. 6. Путь к файлу меньше 259 символов. 7. Путь к файлу равен 260 символов. 8. Путь к файлу больше 260 символов. 9. Корректный путь введен с клавиатуры. 10. Имитировать сбой загрузки (например, с использованием flash-накопителя). 11. Одновременная загрузка нескольких файлов	1. Унификация дизайна по отношению ко всему приложению (цвет, шрифт, высота/ширина). 2. Выравнивание названий загруженных файлов

Таблица 10 – Перечень основных проверок для «радиобаттона»

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

Таблица 11 – Перечень основных проверок для чекбоксов

Functional Test	GUI Test
1. Функциональность: включение/выключение. 2. Наличие дополнительного чекбокса, выставляющего/снимающего все чекбоксы при наличии больше 10 чекбоксов. 3. При переходе на следующую страницу и возвращении назад выбранный чекбокс не должен сбрасываться. 4. Активация путем нажатия как на символ, так и на текст	1. Унификация дизайна по отношению ко всему приложению. 2. Выравнивание расположения чекбокса с соответствующим названием. 3. Изменение чекбокса при наведении курсора. 4. Изменение курсора при наведении на чекбокс

Таблица 12 – Перечень основных проверок для поля со списком

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

Таблица 13 – Перечень основных проверок для меню

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

Таблица 14 – Перечень основных проверок для ссылки

Functional Test	GUI Test
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	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 15 – Перечень основных проверок для таблицы

Functional Test	GUI Test
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	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 16 – Перечень основных проверок для календаря

Functional Test	GUI Test
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)	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 17 – Перечень основных проверок для сообщений

Functional Test	GUI Test
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	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»)

Создание проверок в баг-трекинге.

В качестве примеров предложены две баг-трекинговые системы: Taiga и HacknPlan.

Для работы с Taiga необходимо выполнить следующий алгоритм действий:

1. Перейдите по ссылке https://tree.taiga.io/register и зарегистрируйтесь в системе управления проектами.

2. После регистрации создайте новый проект (рисунок 3).

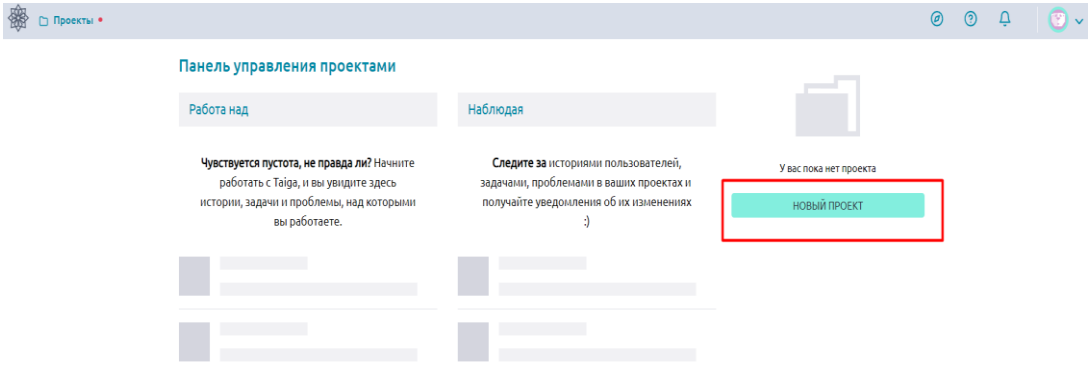

Рисунок 3 – Создание нового проекта

3. Выберите методологию проекта. В примере выбрана методология SCRUM (рисунок 4).

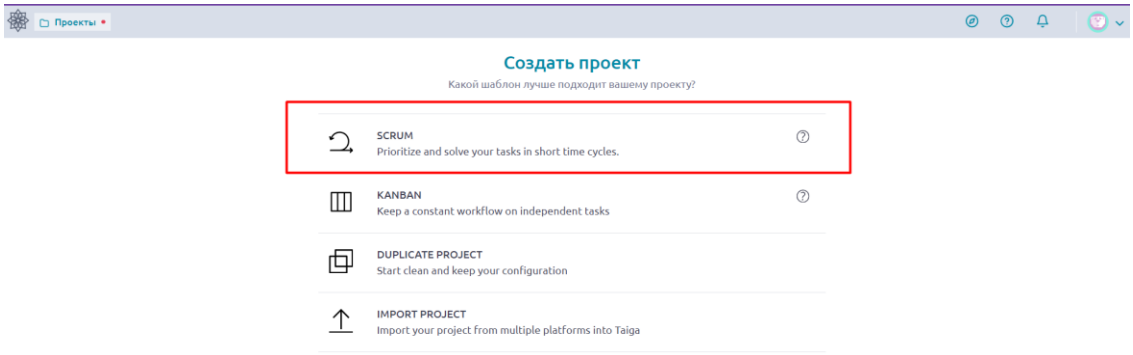

Рисунок 4 – Выбор методологии

4. Укажите название и описание проекта (рисунок 5).

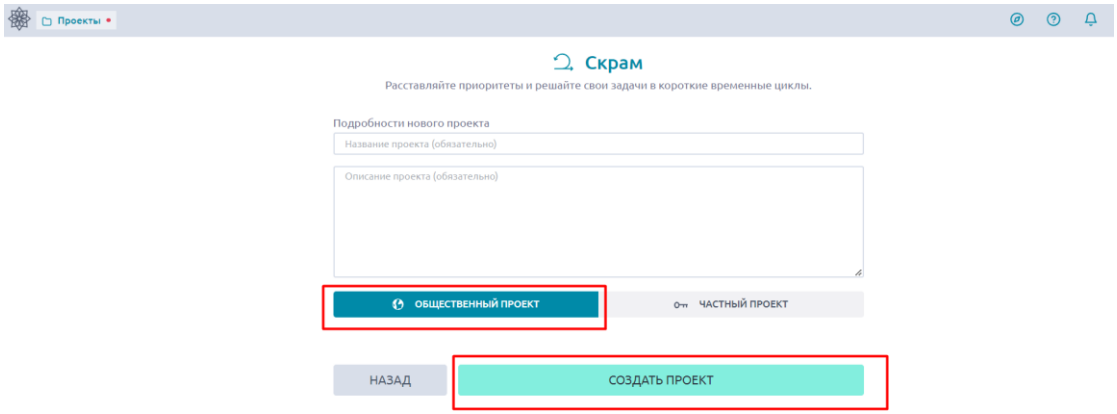

Рисунок 5 – Создание проекта

5. После создания проекта добавьте участников проекта перейдя в раздел Settings (Настройки) – MEMBERS (Участники) – NEW MEMBERS (Новые участники) (рисунок 6).

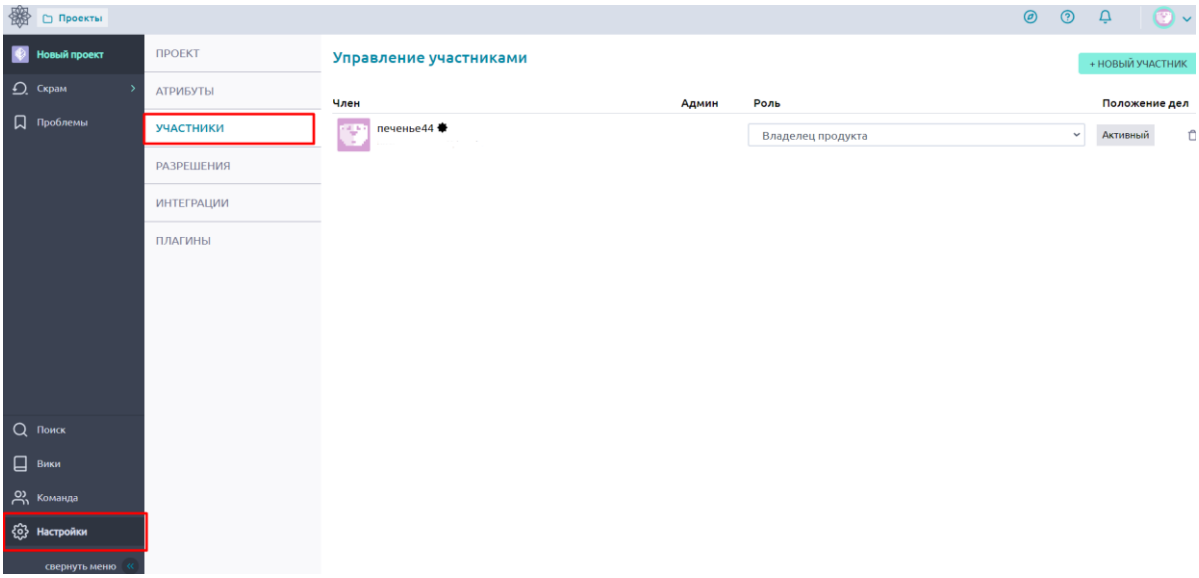

Рисунок 6 – Добавление участников

6. После нажатия на кнопку «+NEW MEMBER» (Новый участник) введите электронную почту участника, который ранее зарегистрировался в системе управления проектами и нажмите на кнопку добавления участника (рисунок 7).

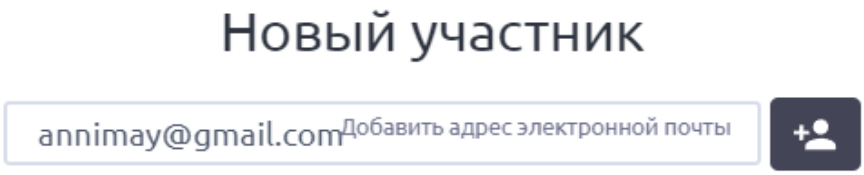

Рисунок 7 – Регистрация нового участника

7. Далее выберите роль и нажмите на кнопку «INVITE» (Пригласить) (рисунок 8).

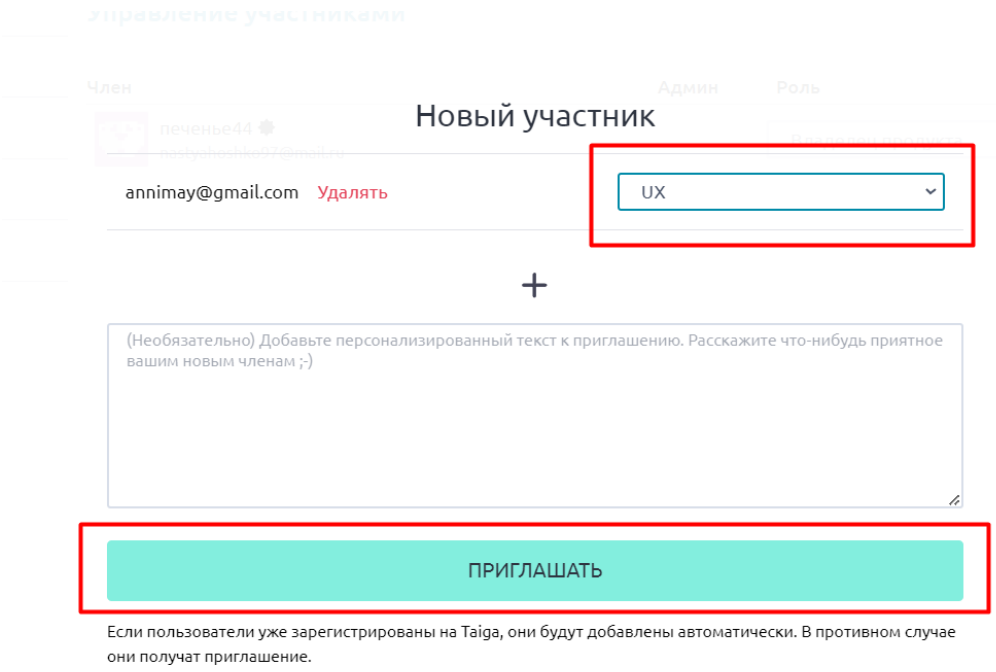

Рисунок 8 – Приглашение нового участника

8. Участник проекта отобразится в разделе «Manage members» (Управление участниками) (рисунок 9).

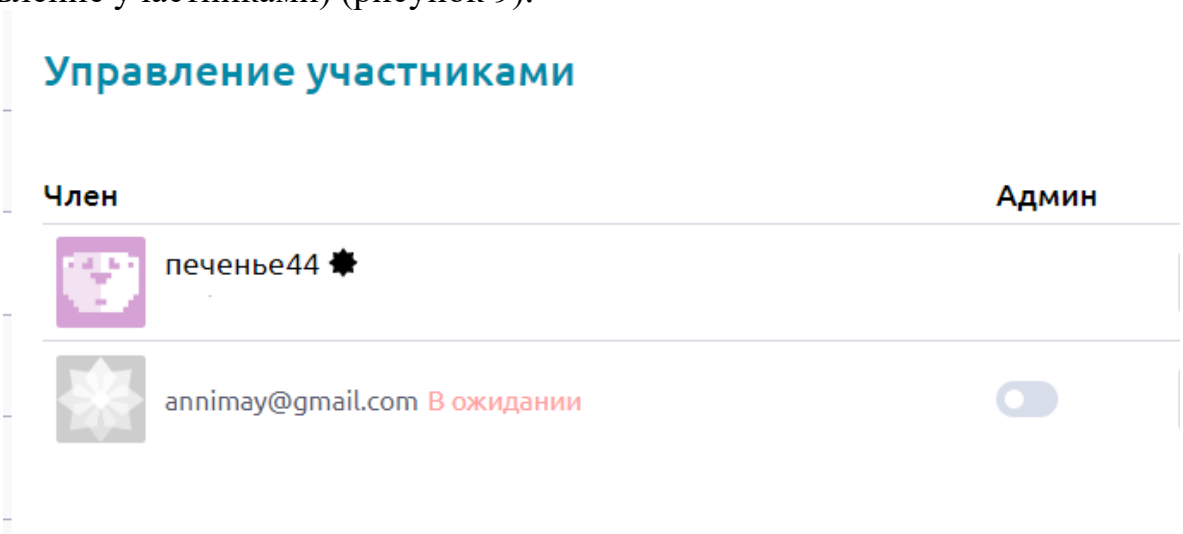

Рисунок 9 – Управление участниками

9. Для написания тест-кейса перейдите в раздел «Issues» (Задачи) и нажмите на кнопку «NEW ISSUE» (Новые задачи) (рисунки 10, 11).

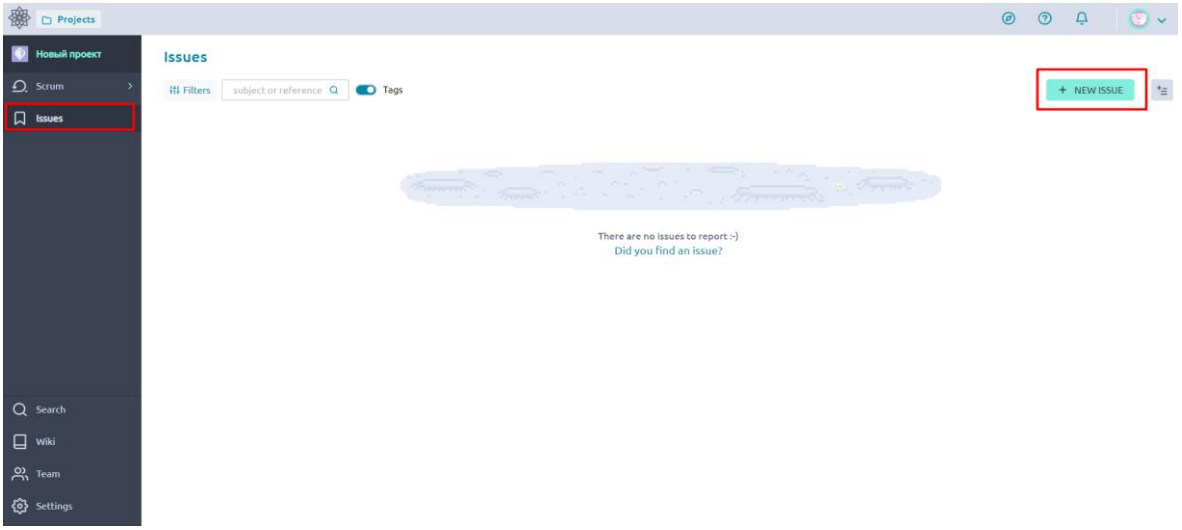

Рисунок 10 – Добавление задачи

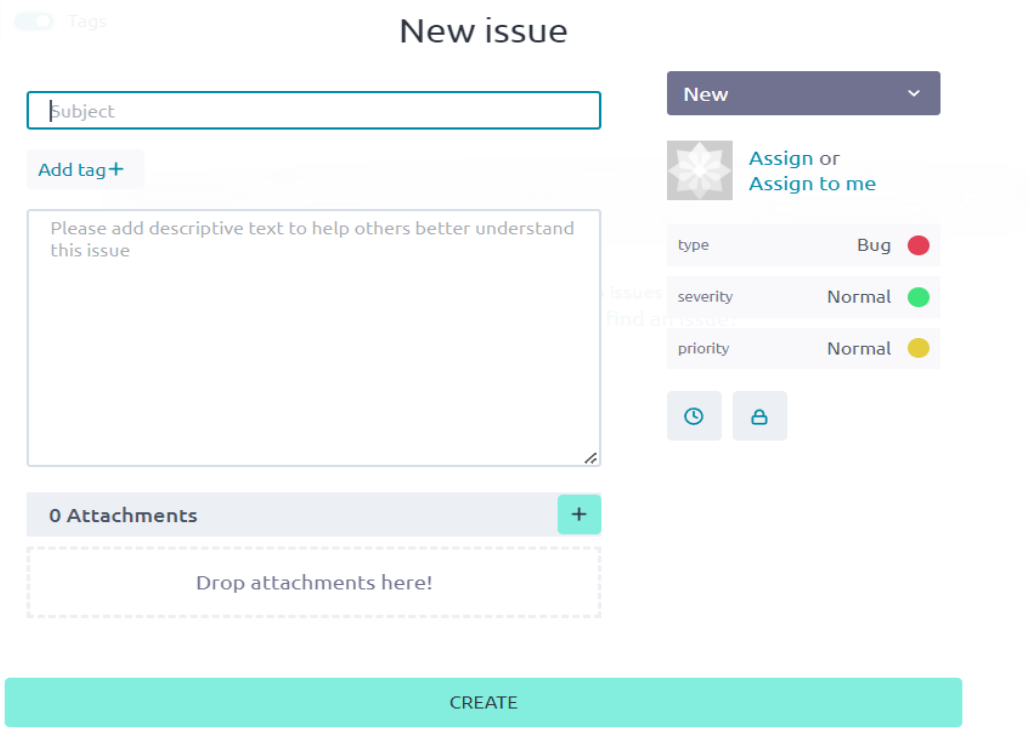

Рисунок 11 – Добавление новой задачи

10. В открывшемся окне (рисунки 12, 13):

– Выберите type Question (тип – «Вопрос»), укажите тип и приоритет.

– Укажите в теге, к какому модулю или подмодулю программы относится проверка.

– В текстовом поле укажите шаги проверки и ожидаемый результат.

– После заполнения всех необходимых полей нажмите на кнопку «CREATE» (Создать).

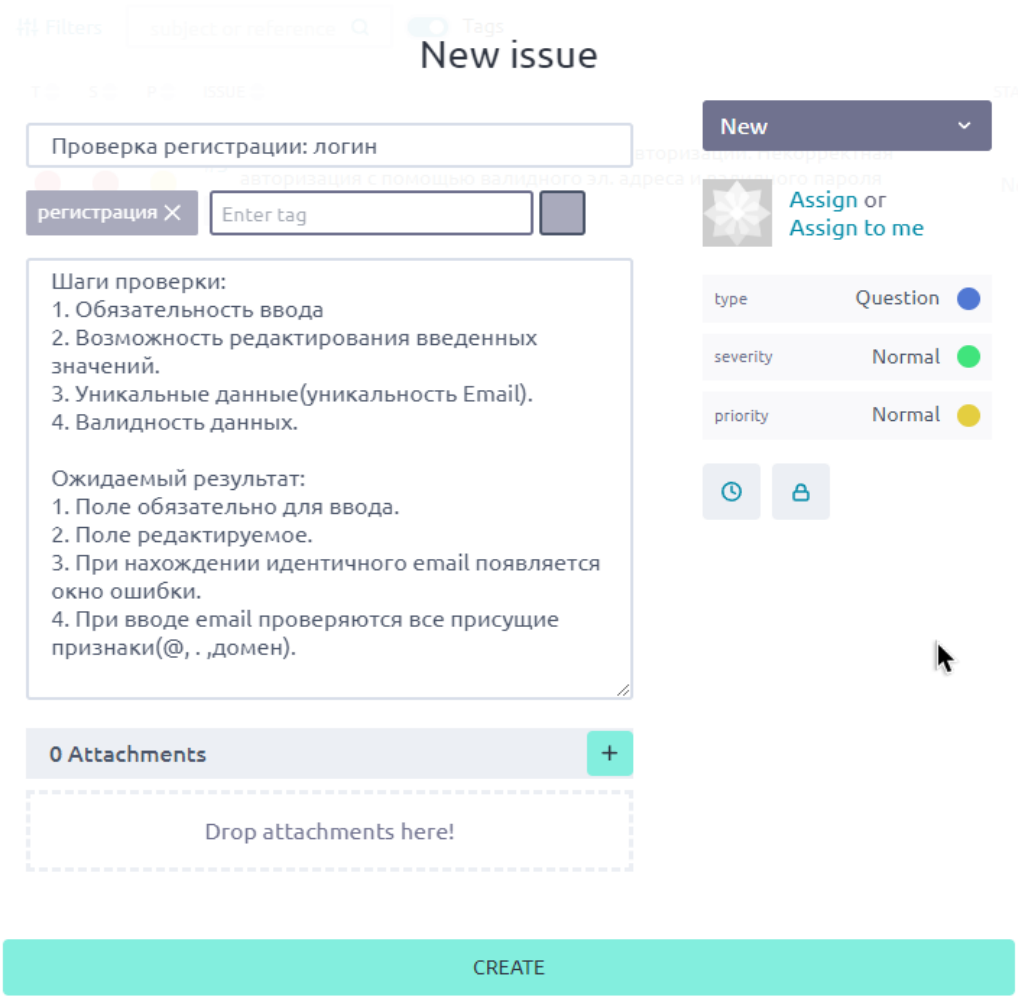

Рисунок 12 – Заполнение окна «NEW ISSUE» (Новые задачи)

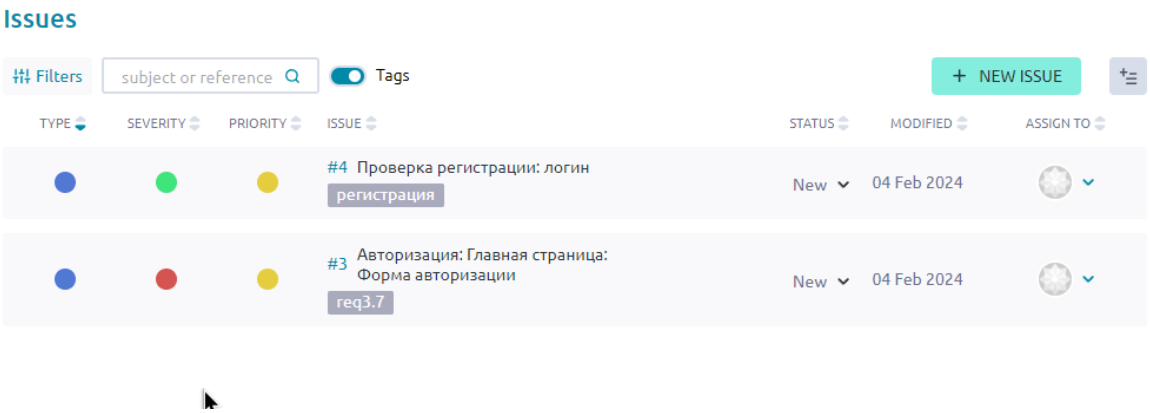

Рисунок 13 – Создание проверки

Для работы с HacknPlan необходимо выполнить следующее:

1. Для работы с баг-трекинговой системой перейдите по ссылке: hacknplan.com

2. Далее нажмите на кнопку Try now (Попробовать сейчас) (рисунок 14).

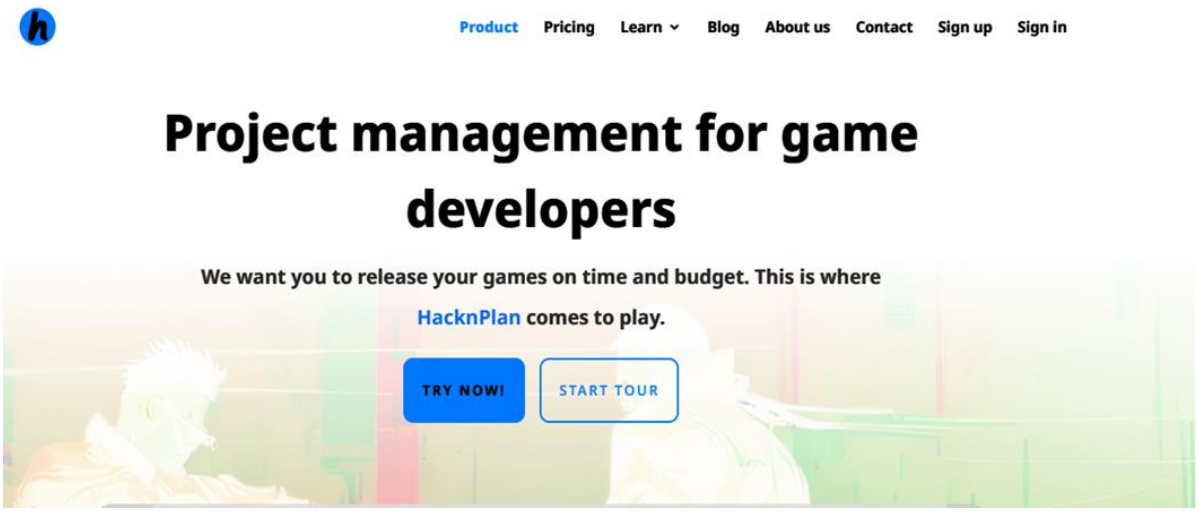

Рисунок 14 – Главная страница системы HacknPlan

3. Заполните форму регистрации (рисунок 15).

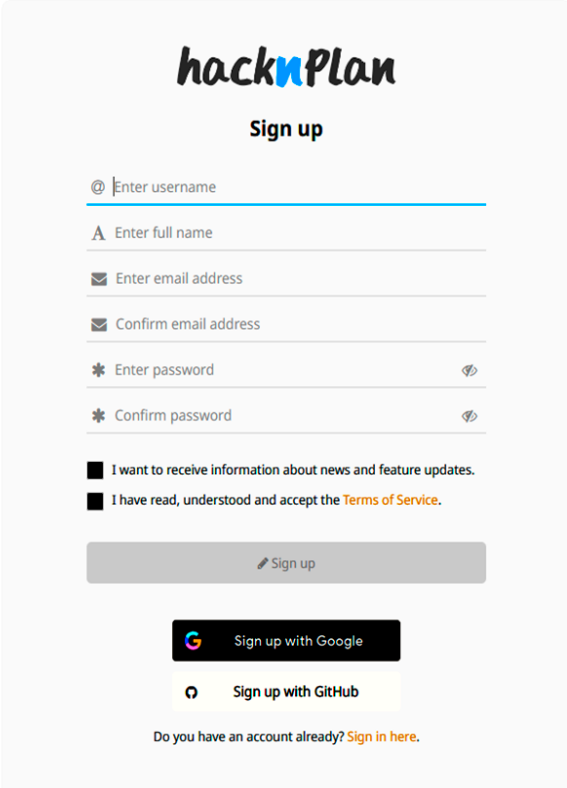

Рисунок 15 – Окно регистрации системы

4. Далее необходимо выполнить настройку профиля (рисунок 16).

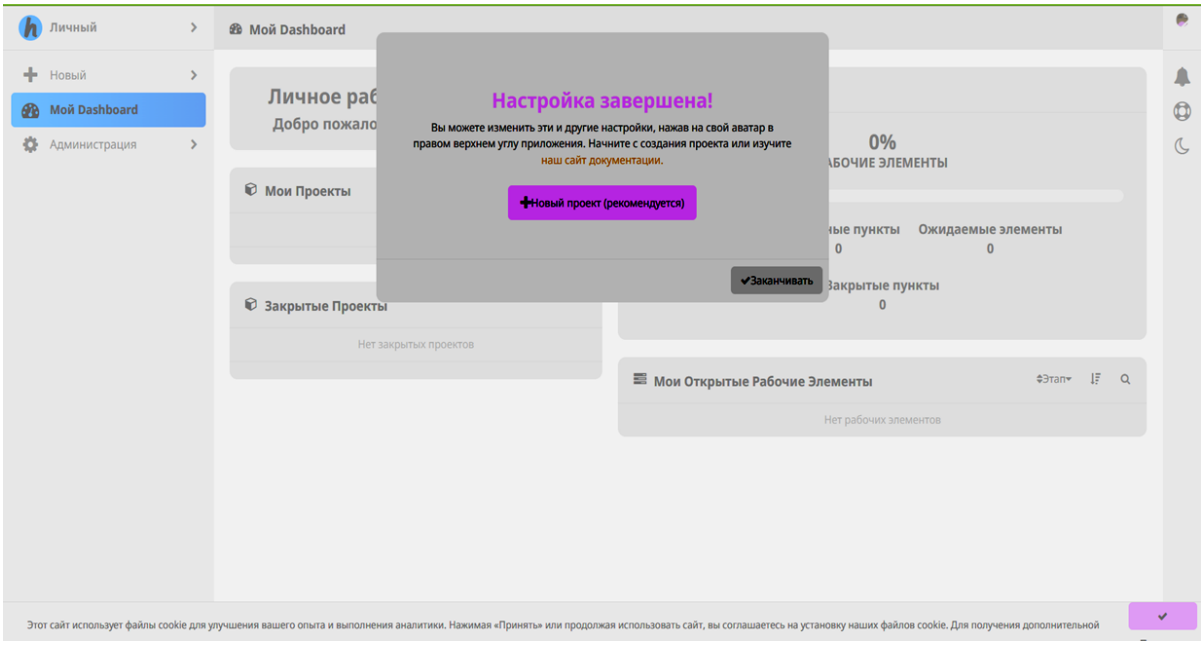

Рисунок 16 – Выполнение настройки профиля

5. Создайте проект (выберите тип проекта – простой проект) и добавьте участников проекта посредством кнопки «Добавить пользователей» (рисунки 17, 18).

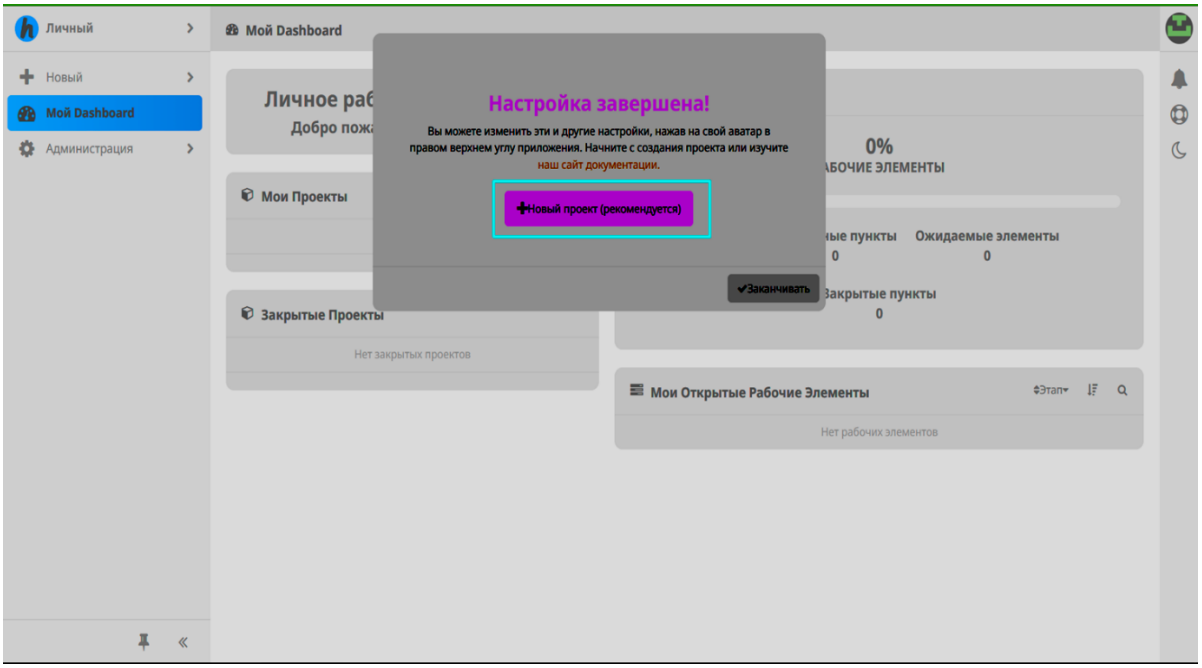

Рисунок 17 – Создание нового проекта

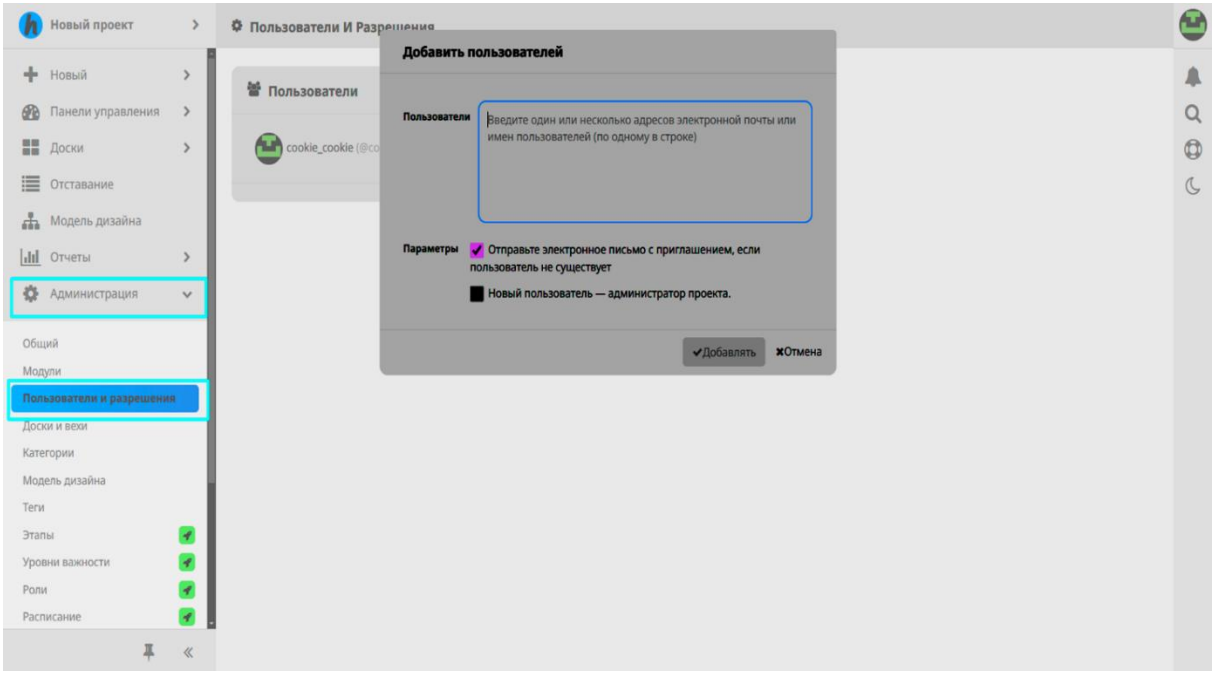

Рисунок 18 – Добавление новых участников проекта

6. После создания проекта необходимо выбрать его во вкладке «Моя доска» (рисунок 19).

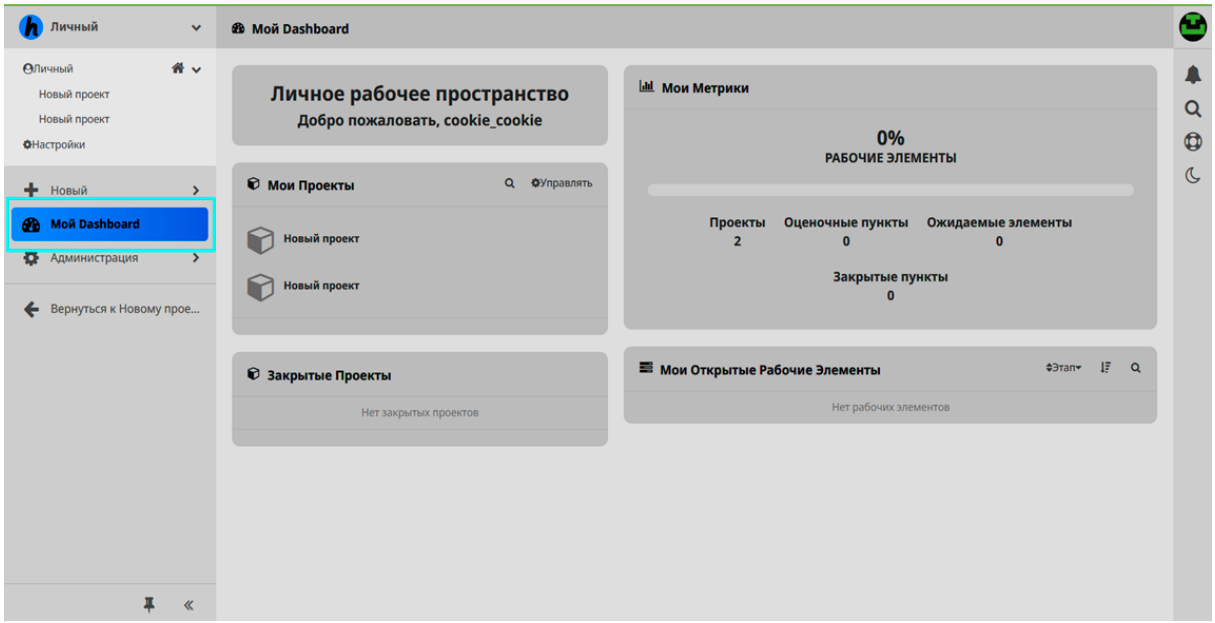

Рисунок 19 – Вкладка «Моя доска»

7. В рабочем пространстве проекта создайте новую задачу (укажите название и нажмите кнопку «Создать» (рисунки 20, 21)).

Рисунок 20 – Рабочее пространство проекта

Риссунок 21 – Добавление новой задачи

8. Выберите созданную задачу в окне описания, внесите шаги проверок, а в поле подзадачи – ожидаемый результат (рисунок 22).

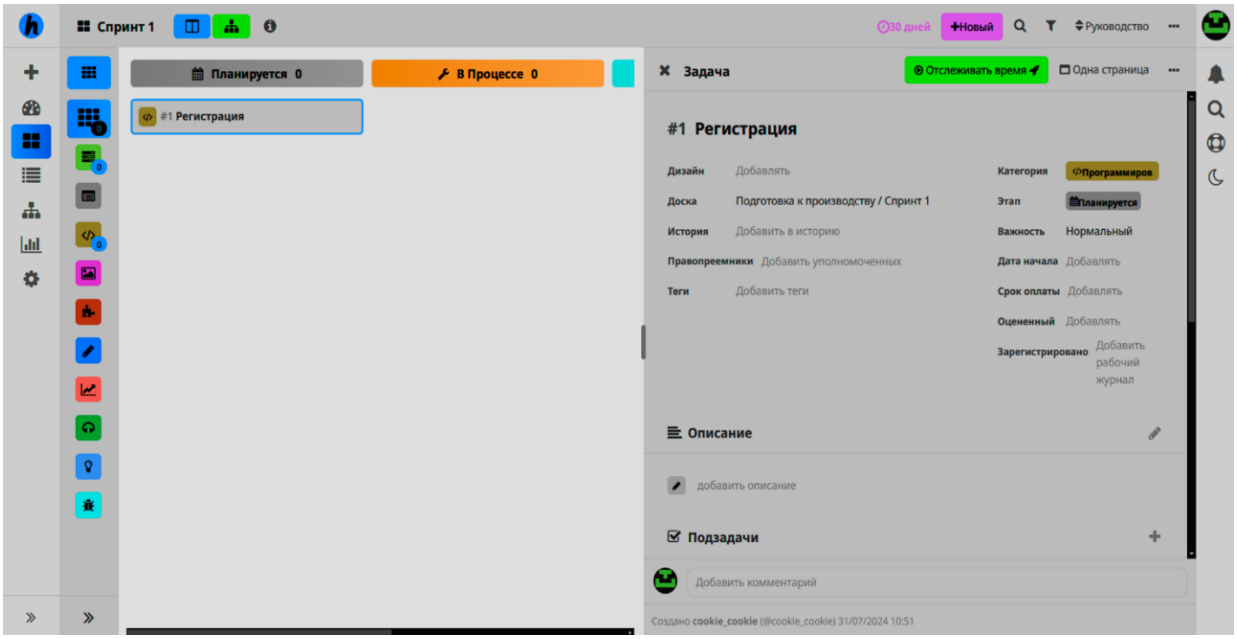

Рисунок 22 – Редактирование задачи

Практическое задание:

1. Получить у преподавателя спецификацию с требованиями к веб-приложению.
2. В зависимости от сложности бизнес-логики веб-приложения выбрать наиболее подходящий вид рабочей тестовой документации (Acceptance Sheet, Test Survey, Test Cases либо баг-трекинговую систему).
3. Анализируемое веб-приложение разбить на модули и подмодули.
4. Разработать рабочую тестовую документацию для всех модулей и подмодулей веб-приложения.
5. Указать номер тестируемой сборки, название приложения, тип выполняемой тестовой активности, период времени тестирования, Ф. И. О. 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 № 4

Поиск и документирование дефектов. Документирование результатов тестирования

Цель: протестировать веб-приложение и описать найденные дефекты, составить итоговый отчет о результатах тестирования веб-приложения.

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 баг-трекинговой системе (например, Taiga, HacknPlan, JIRA, Bugzilla, Mantis, Redmine и др.) или в предварительно созданном в программной среде Microsoft Excel файле (пример приведен в таблице 18).

Таблица 18 – Пример описания дефекта

№	Название дефекта	Важность	Алгоритм воспроизведения	Фактический результат	Ожидаемый результат	Приложение	Примечание
9	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»	Не соответствует ожидаемому	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 баг-трекинговых 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 – предоставить краткую и в то же 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 на эту кнопку 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 19): 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 19 – Степени критичности дефектов

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

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, видеоошибки и т. д. Attachment является вспомогательным 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 23).

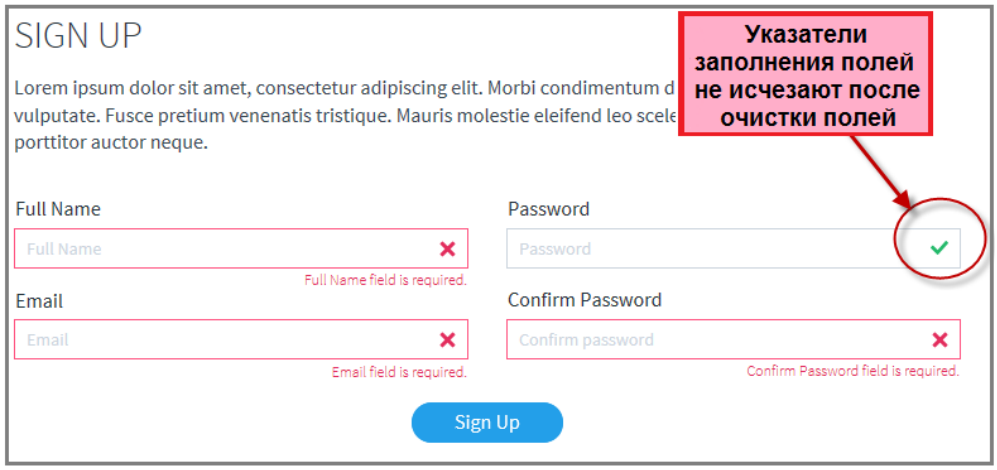

Рисунок 23 – Пример скриншота, поясняющего обнаруженный дефект

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 веб-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.

Ниже 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. Если пренебрегать 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.

Работа с баг-трекерами.

В качестве примера приведены баг-трекинговые системы: Taiga и HacknPlan.

Пример работы с системой Taiga (рисунок 24):

1. При нахождении дефекта внутри тест-кейса меняем тип с «Question» на «Issue».

2. Выберите «Severity» и «Priority».

3. В текстовое поле напишите фактический результат.

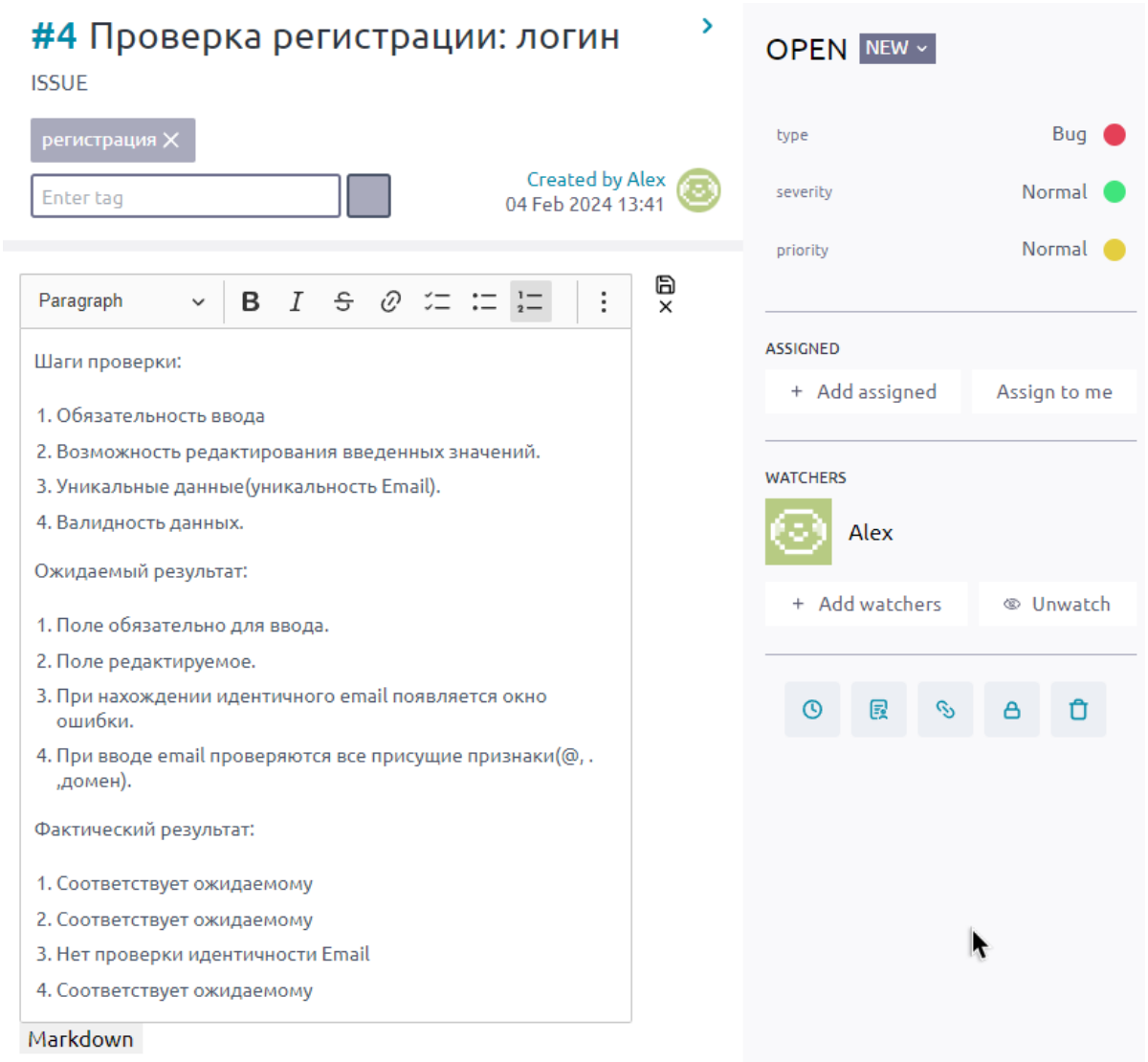

Рисунок 24 – Работа с баг-трекингом Taiga

4. В комментариях к тест-кейсу необходимо написать шаги воспроизведения дефекта (рисунок 25).

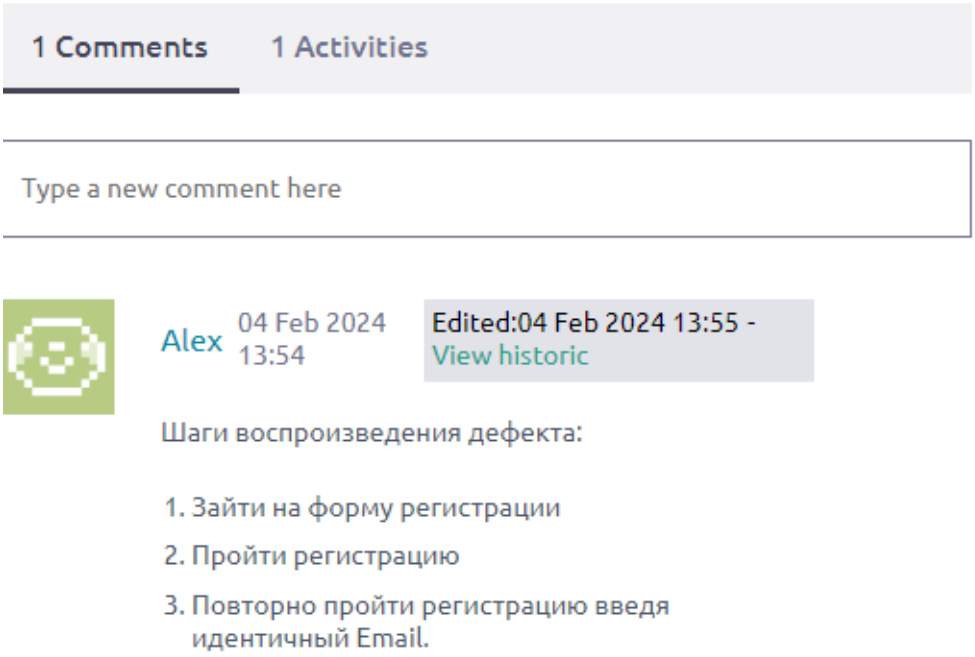

Рисунок 25 – Шаги воспроизведения дефекта

5. Добавьте скриншот дефекта в поле Attachments (Предложения) (рисунок 26).

Рисунок 26 – Добавление скриншота дефекта

Пример работы с баг-трекинговой системой HacknPlan.

При нахождении дефекта необходимо (рисунок 27):

1. Изменить категорию на «Дефект».
2. В поле «Комментарий» внести название и шаги воспроизведения дефекта.
3. Приложить скриншот в поле вложения.

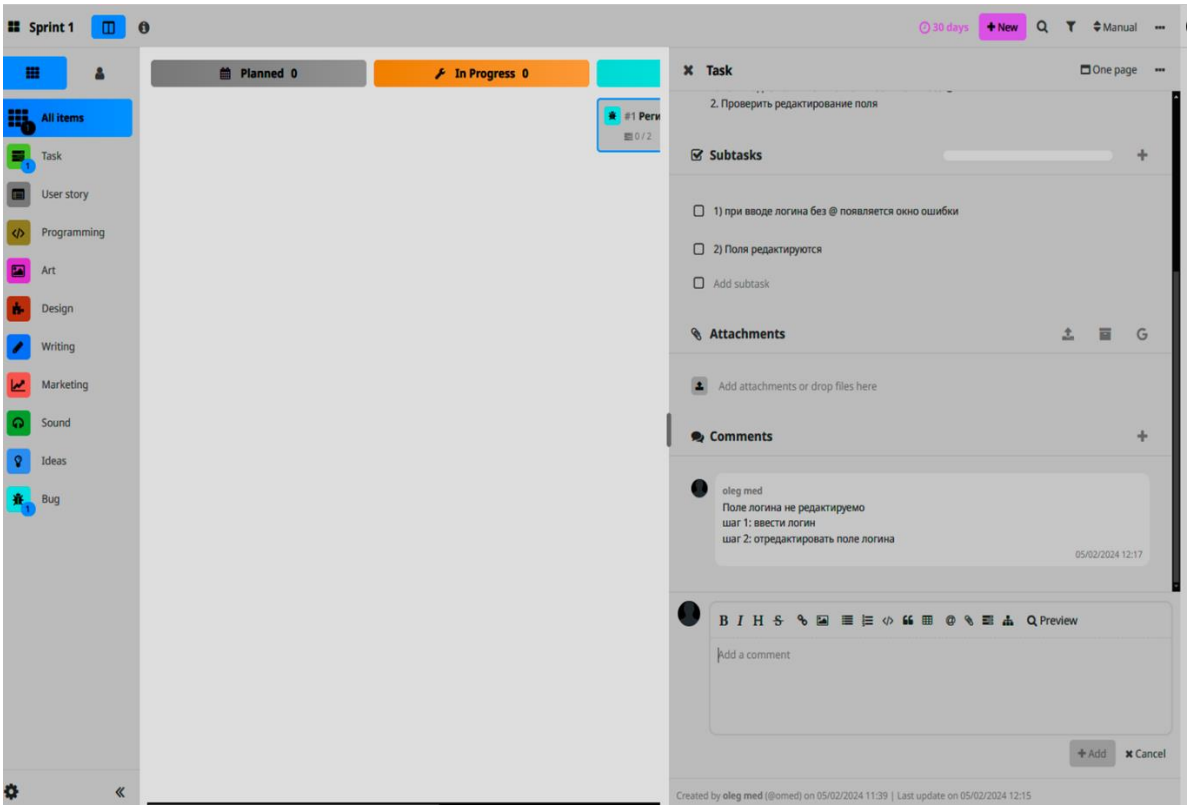

Рисунок 27 – Работа с дефектом

Итоговый отчет о качестве проверенного функционала является неотъемлемой частью работы, которую каждый тестировщик должен выполнить по завершении 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.
10. Пример итогового отчета приведен на рисунке 28.

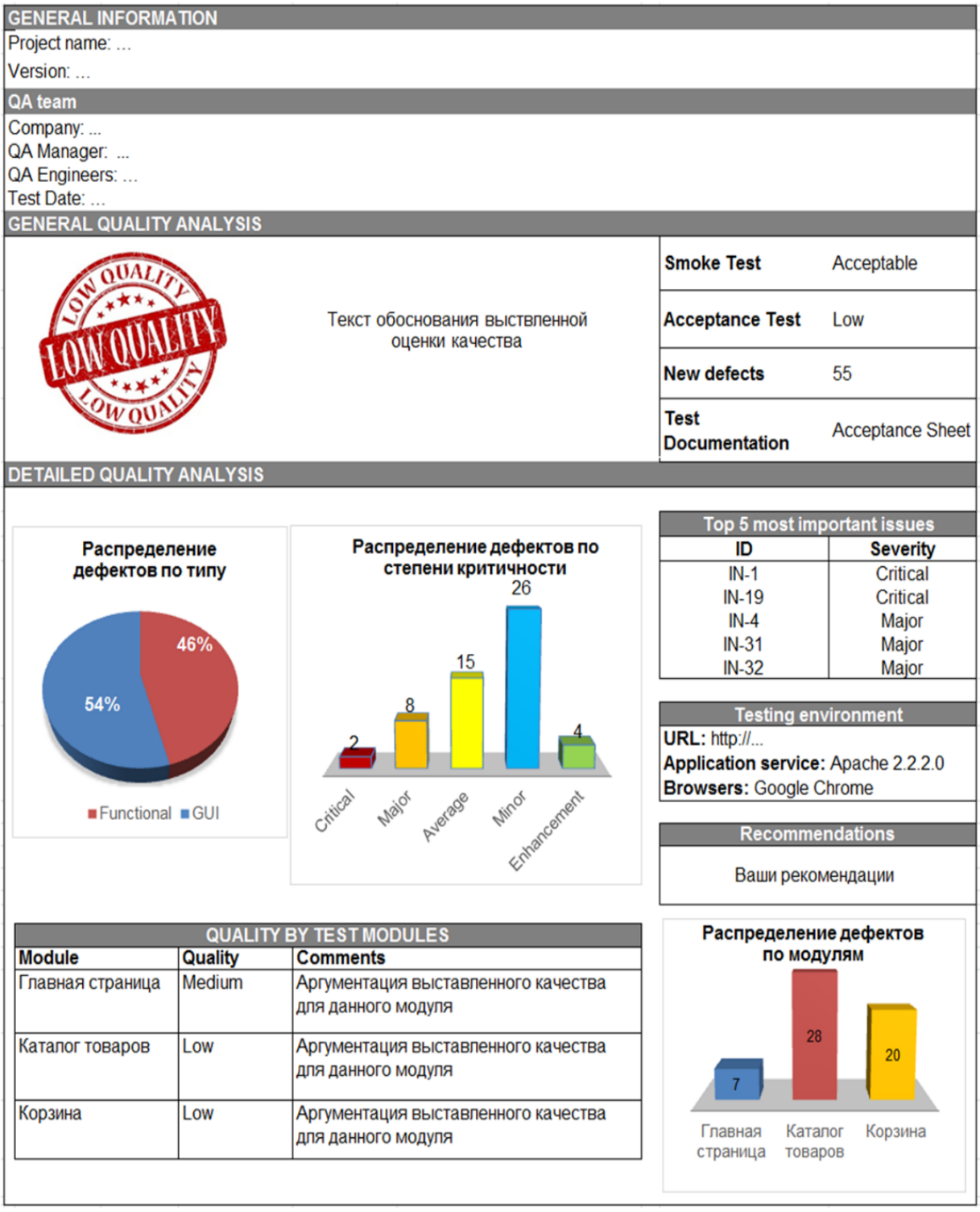

Рисунок 28 – Пример итогового отчета о результатах тестирования

Ниже 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).

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:

$$T^{SP} = \frac{T^{Success}}{T^{Total}} \cdot 100\ \%, \quad (1)$$

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

$$R^{SimpleCoverage} = \frac{R^{Covered}}{R^{Total}} \cdot 100\ \%, \quad (2)$$

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

T^{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 17).

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

Если это релизная или предрелизная 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 отдельно проверенного функционала может быть высокое (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 высокое (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 предрелизной, то любое ухудшение качества является критичным и важно это обозначить.

Практическое задание:

1. Сформулировать по два возможных дефекта на каждый уровень Severity (Critical, Major, Average, Minor, Enhancement) для выбранного объекта реального мира.
2. Описать по одному дефекту на каждый уровень Severity (Critical, Major, Average, Minor, Enhancement) для выбранного объекта реального мира.
3. Протестировать веб-приложение в соответствии с составленной ранее тестовой документацией.
4. Описать все найденные дефекты в отчете о дефектах в среде Microsoft Excel.
5. В отчете о дефектах указать номер тестируемой сборки, название приложения, период времени тестирования, Ф. И. О. тестировщика, тестовое окружение (операционная система, браузер).
6. Для каждого дефекта указать его порядковый номер, заголовок, важность, алгоритм воспроизведения, фактический результат, ожидаемый результат, приложение, примечание.
7. Для каждого дефекта обязательно сделать скриншоты.
8. В рабочую тестовую документацию внести результаты тестирования с указанием напротив соответствующей проверки степени критичности обнаруженного дефекта, его номера и заголовка. Составить итоговый отчет по результатам тестирования веб-приложения.
9. Указать общую информацию о тестируемом продукте (название, номер сборки, виды выполненных тестов, количество обнаруженных дефектов, вид рабочей тестовой документации).
10. Указать, кто и когда тестировал программный продукт.
11. Описать тестовое окружение (ссылку на веб-приложение, браузер).
12. Указать общую оценку качества протестированного приложения и подробно ее обосновать.
13. Графически (в виде круговой диаграммы) отразить процентное соотношение дефектов GUI и функциональных дефектов.
14. Графически (в виде столбчатой диаграммы) отразить распределение дефектов по различным степеням критичности.
15. Графически (в виде столбчатой диаграммы) отразить распределение дефектов по модулям.
16. Произвести детальный анализ качества всех модулей протестированного приложения с аргументацией выставленных уровней качества.

17. Привести список пяти наиболее критичных дефектов.

18. Сформулировать рекомендации по улучшению качества программного продукта.

19. 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 веб-приложения.

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

5. Выводы по работе.

6. Итоговый отчет о результатах тестирования веб-приложения.

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?

12. Какова структура итогового отчета о результатах тестирования?

13. Что содержится в разделе «Общая информация»?

14. Что содержится в разделе «Тестовое окружение»?

15. Как выставляется общая оценка качества приложения?

16. Как обосновать выставленную оценку качества?

17. Что такое метрика в тестировании?

18. Приведите примеры прямых метрик.

19. Приведите примеры расчетных метрик.

20. Для чего используется графическое представление результатов тестирования в итоговом отчете?

21. Что содержится в разделе «Детализированный анализ качества»?

22. Что содержится в разделе «Рекомендации»?

Лабораторная работа № 5

Автоматизированное тестирование

Цель: разработать автоматизированные тесты с использованием библиотеки Selenium WebDriver.

План занятия:

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

Теоретические сведения

Selenium WebDriver – это пакет программ с открытым исходным кодом. Они помогают тестировать программное обеспечение, а также администрировать его: локально или при помощи интернета.

Поиск элементов с помощью Selenium. Для поиска элементов на странице в Selenium WebDriver используются несколько стратегий, позволяющие искать по атрибутам элементов, текстам в ссылках, CSS-селекторам и XPath-селекторам. Существуют следующие методы поиска элементов:

find_element_by_id – поиск по уникальному атрибуту *id* элемента. Если разработчики проставляют всем элементам в приложении уникальный *id*. Данный метод используется чаще всего, т. к. является наиболее стабильным;

find_element_by_css_selector – поиск элемента с помощью правил на основе CSS. Это универсальный метод поиска, т. к. большинство веб-приложений использует CSS для верстки и задания оформления страницам. Является альтернативой *find_element_by_id* из-за отсутствия *id* у элементов;

find_element_by_xpath – поиск с помощью языка запросов XPath, позволяет выполнять очень гибкий поиск элементов;

find_element_by_name – поиск по атрибуту name элемента;

find_element_by_tag_name – поиск элемента по названию тега элемента;

find_element_by_class_name – поиск по значению атрибута class;

find_element_by_link_text – поиск ссылки на странице по полному совпадению;

find_element_by_partial_link_text – поиск ссылки на странице, если текст селектора совпадает с любой частью текста ссылки.

Например, необходимо найти кнопку со значением *id=“submit_button”*:

```
from selenium import webdriver
browser = webdriver.Chrome()
browser.get(“http://suninjuly.github.io/simple_form_find_task.html”)
button = browser.find_element_by_id(“submit”)
```

Если страница загрузилась, но дальше ничего не происходит, необходимо вернуться обратно в консоль, в которой был запущен скрипт. Скорее всего, это ошибка «*NoSuchElementExceptin»*. Она будет выглядеть следующим образом:

selenium.common.exceptions.NoSuchElementException: Message: no such element: Unable to locate element: {"method":"id","selector":"submit"}

Ошибка очевидна: неправильно указан локатор – значит кнопки с таким *id* на странице нет.

Решение: необходимо исправить локатор, чтобы код проходил без ошибок:

```
from selenium import webdriver
browser = webdriver.Chrome()
browser.get("http://suninjuly.github.io/simple_form_find_task.html")
button = browser.find_element_by_id("submit_button")
```

Термин «локатор» подразумевает стратегию поиска и значения, по которым должен выполняться поиск. Например, можно искать по локатору *By.ID* со значением *"send_button"*.

Существует второй способ для поиска элементов с помощью универсального метода *find_element()* и полей класса *By* из библиотеки Selenium.

Пример:

```
from selenium import webdriver
from selenium. webdriver.common.by import By
browser = webdriver.Chrome()
browser.get("http://suninjuly.github.io/simple_form_find_task.html")
button = browser.find_element(By.ID, "submit_button")
```

Можно использовать те же стратегии поиска, что и в первом способе. Второй способ более удобен для оформления архитектуры тестовых сценариев с помощью подхода *Page Object Model*.

Поля класса *By*, которые можно использовать для поиска:

By.ID – поиск по уникальному атрибуту *id* элемента;

By.CSS_SELECTOR – поиск элементов с помощью правил на основе CSS;

By.XPATH – поиск элементов с помощью языка запросов XPath;

By.NAME – поиск по атрибуту name элемента;

By.TAG_NAME – поиск по названию тега;

By.CLASS_NAME – поиск по атрибуту class элемента;

By.LINK_TEXT – поиск ссылки с указанным текстом. Текст ссылки должен быть точным совпадением;

By.PARTIAL_LINK_TEXT – поиск ссылки по частичному совпадению текста.

Бывают ситуации, когда на странице будет несколько элементов, подходящих под заданные параметры поиска. В этом случае WebDriver вернет только первый элемент, который встретит во время поиска по HTML. Если нужен не первый, а второй или следующие элементы, нужно либо задать более точный селектор для поиска, либо использовать методы *find_elements_by*.

Работа с браузером в Selenium. После запуска примеров скриптов браузер не всегда закрывается после выполнения кода. Поэтому обратите внимание на то, что необходимо явно закрывать окно браузера в коде при помощи команды *browser.quit().* Каждый раз при открытии браузера *browser = webdriver.Chrome()* в системе создается процесс, который останется висеть, если вручную закрывать окно браузера. Чтобы не остаться без оперативной памяти после запуска нескольких скриптов, всегда необходимо добавлять к своим скриптам команду закрытия:

```
from selenium import webdriver
from selenium.  webdriver.common.by import By
link = “http://suninjuly.github.io/simple_form_find_task.html”
browser = webdriver.Chrome()
browser.get(link)
button = browser.find_element(By.ID, “submit_button”)
button.click()
# закрываем браузер после всех манипуляций
browser.quit()
```

Важно отметить разницу между двумя командами: *browser.close()* и *browser.quit()*.

browser.close() закрывает текущее окно браузера. Это значит, что если скрипт вызвал всплывающее окно или открыл что-то в новом окне или вкладке браузера, то закроется только текущее окно, а все остальные останутся висеть.

В свою очередь, *browser.quit()* закрывает все окна, вкладки и процессы веб-драйвера, запущенные во время тестовой сессии.

Вопрос: если скрипт не дойдет до выполнения этого финального шага, а упадет с ошибкой где-то раньше?

Для того чтобы гарантировать закрытие, даже если произошла ошибка в предыдущих строках, проще всего использовать конструкцию *try/finally*:

```
from selenium import webdriver
from selenium. webdriver.common.by import By
link = “http://suninjuly.github.io/simple_form_find_task.html”
try:
    browser = webdriver.Chrome()
    browser.get(link)
    button = browser.find_element(By.ID, “submit”)
    button.click()
finally:
    # закрываем браузер после всех манипуляций
    browser.quit()
```

Важно понимать только то, что даже если в коде внутри блока *try* произойдет какая-то ошибка, то код внутри блока *finally* выполнится в любом случае. Поэтому желательно добавлять такую обработку ко всем скриптам при выполнении задач этого и следующего модулей.

В качестве примеров рассмотрим все перечисленные поиски элементов с помощью Selenium.

Поиск элементов с помощью Selenium:

1. Перейдите по ссылке (https://suninjuly.github.io/simple_form_find_task.html) и заполните форму на этой странице с помощью Selenium. Если все сделано правильно, то вы увидите окно с проверочным кодом. Это число вам нужно ввести в качестве ответа в этой задаче. Обратите внимание, что время для ввода данных ограничено. Однако благодаря Selenium вы сможете выполнить задачу до того, как время истечет.

2. Для решения этой задачи подготовлен шаблон кода, в который следует только подставить нужные значения для поиска вместо слов value1, value2 и т. д. Обратите внимание, что значения нужно заключать в кавычки, т. к. они должны передаваться в виде строки.

```
from selenium import webdriver
import time
link = "http://suninjuly.github.io/simple_form_find_task.html"
try:
    browser = webdriver.Chrome()
    browser.get(link)
    input1 = browser.find_element_by_tag_name(value1)
    input1.send_keys("Ivan")
    input2 = browser.find_element_by_name(value2)
    input2.send_keys("Petrov")
    input3 = browser.find_element_by_class_name(value3)
    input3.send_keys("Smolensk")
    input4 = browser.find_element_by_id(value4)
    input4.send_keys("Russia")
    button = browser.find_element_by_css_selector("button.btn")
    button.click()
finally:
    # успеваем скопировать код за 30 секунд
    time.sleep(30)
    # закрываем браузер после всех манипуляций
    browser.quit()

# не забываем оставить пустую строку в конце файла
```

Системы UNIX/Linux ожидают пустую строку в конце файла, если в вашем скрипте ее не будет, то последняя строчка, содержащая код, может не выполниться.

3. Создайте файл *lesson5_step4.py* (обратите внимание на расширение *.py*) и вставьте туда шаблон кода. Подберите селекторы и запустите из командной строки: *python lesson5_step4.py.*

Поиск элемента по тексту в ссылке с помощью метода find_element_by_link_text: link = browser.find_element_by_link_text(text).

В качестве аргумента в метод передается такой текст, ссылку с которым нужно найти. Это тот самый текст, который содержится между открывающим и закрывающим тегом <a> вот тут .

Допустим, на странице https://www.degreesymbol.net/ необходимо найти ссылку с текстом “Degree symbol examples” и перейти по ней. Если хотим найти элемент по полному соответствию текста, тогда код будет иметь такой вид:

link = browser.find_element_by_link_text(“Degree symbol examples”)

link.click()

А если хотим найти элемент со ссылкой по подстроке, то нужно написать следующий код:

link = browser.find element by partial link text(“examples”)

link.click()

Обычно поиск по подстроке чуть более удобный и гибкий, но с ним надо быть вдвойне аккуратными и проверять, что находится нужный элемент.

Попробуем выполнить поиск.

На указанной ниже странице нужно найти зашифрованную ссылку и кликнуть по ней:

1. Добавьте в самый верх своего кода *import math.*
2. Добавьте команду в код, которая откроет страницу: http://suninjuly.github.io/find_link_text.
3. Добавьте команду, которая найдет ссылку с текстом. Текст ссылки, который нужно найти, зашифрован формулой

*str(math.ceil(math.pow(math.pi, math.e)*10000))*

4. Добавьте команду для клика по найденной ссылке: она перенесет вас на форму регистрации.
5. Заполните скриптом форму так же, как вы делали в предыдущем шаге урока.
6. После успешного заполнения вы получите код – отправьте его в качестве ответа на это задание.

Важно отметить, что поиск по тексту ссылки бывает очень удобным, т. к. часто тексты меняются реже, чем атрибуты элементов. Но лучше избегать такого метода поиска. Например, если приложение имеет несколько языков интерфейса, тесты будут проходить только с определенным языком интерфейса.

Поиск всех необходимых элементов с помощью find_elements_by.

Метод *find_element_by* возвращает только первый из всех элементов, которые подходят под условия поиска. Иногда возникает ситуация, когда есть несколько одинаковых по сути объектов на странице, например иконки товаров в корзине интернет-магазина. В тесте нужно проверить, что отображаются все выбранные для покупки товары. Для этого существуют методы *find_elements_by*, которые в отличие от *find_element_by* вернут список всех найденных элементов по заданному условию. Проверив длину списка, можно удостовериться, что в

корзине отобразилось правильное количество товаров. Пример кода (код приведен только для примера, сайта fake-shop.com не существует):

```
# подготовка для теста
# открываем страницу первого товара
# данный сайт не существует, этот код приведен только для примера
browser.get("https://fake-shop.com/book1.html")
# добавляем товар в корзину
add_button = browser.find_element_by_css_selector(".add")
add_button.click()
# открываем страницу второго товара
browser.get("https://fake-shop.com/book2.html")
# добавляем товар в корзину
add_button = browser.find_element_by_css_selector(".add")
add_button.click()
# тестовый сценарий
# открываем корзину
browser.get("https://fake-shop.com/basket.html")
# ищем все добавленные товары
goods = browser.find_elements_by_css_selector(".good")
# проверяем, что количество товаров равно 2
assert len(goods) == 2
```

Набор стратегий здесь такой же, как и в случае с ***find_element_by***:

*find_element**s**_by_css_selector;*
*find_element**s**_by_xpath;*
*find_element**s**_by_name;*
*find_element**s**_by_tag_name;*
*find_element**s**_by_class_name;*
*find_element**s**_by_link_text;*
*find_element**s**_by_partial_link_text.*

Также для поиска нескольких элементов можно использовать универсальный метод *find_elements* вместе с атрибутами класса *By*:

```
from selenium. webdriver.common.by import By
driver.find_elements(By.CSS_SELECTOR, "button.submit")
```

Важно обратить внимание на разницу в результатах, которые возвращают методы *find_element* и *find_elements*. Если первый метод не смог найти элемент на странице, то он вызовет ошибку *NoSuchElementException*, которая прервет выполнение кода. Второй метод всегда возвращает валидный результат: если ничего не было найдено, то он вернет пустой список и программа перейдет к выполнению следующего шага в коде.

Попробуем выполнить поиск с использованием метода *find_elements_by*.

В этой задаче нужно заполнить форму (http://suninjuly.github.io/huge_form.html). С ее помощью отдел маркетинга компании N захотел собрать подробную информацию о пользователях своего продукта. В награду за заполнение формы

становится доступен код на скидку. Но маркетологи явно переусердствовали, добавив в форму 100 обязательных полей и ограничив время на ее заполнение.

1. Используйте WebDriver и подходящий метод *find_elements_by*. Введите полученный код в качестве ответа к этой задаче.

2. Используйте приведенный ниже шаблон: в цикле *for* можно последовательно взять каждый элемент из найденного списка текстовых полей и отправить произвольный текст в каждое поле. Если скрипт не успевает заполнить форму, выберите текст покороче.

```
from selenium import webdriver
import time
try:
    browser = webdriver.Chrome()
    browser.get("http://suninjuly.github.io/huge_form.html")
    elements = browser.find_elements_by_ * (value)
    for element in elements:
        element.send_keys("Мой ответ")
    button = browser.find_element_by_css_selector("button.btn")
    button.click()
finally:
    # успеваем скопировать код за 30 секунд
    time.sleep(30)
    # закрываем браузер после всех манипуляций
    browser.quit()

# не забываем оставить пустую строку в конце файла
```

Поиск элемента по XPath

На странице http://suninjuly.github.io/find_xpath_form находим такую же форму регистрации, как в пункте при «*Работе с браузером в Selenium»*. Но сработает только кнопка с текстом «Submit», и задача нажать в коде именно на нее.

Далее необходимо воспроизвести следующие шаги:

1. В коде из пункта «*Поиск элементов с помощью Selenium»* замените ссылку на http://suninjuly.github.io/find_xpath_form.

2. Подберите уникальный XPath-селектор так, чтобы он находил только кнопку с текстом «Submit». XPath можете формулировать как угодно (по тексту, по структуре, по атрибутам) – главное, чтобы он работал.

3. Модифицируйте код из пункта при *работе с браузером в Selenium* таким образом, чтобы поиск кнопки происходил с помощью XPath.

4. Запустите код.

Если был подобран правильный селектор и все прошло хорошо, то получите код, который нужно отправить в качестве ответа на это задание.

Уникальность селекторов.

Идеальный селектор – это такой селектор, который позволяет найти только один искомый элемент на странице. Благодаря уникальным селекторам тесты становятся стабильнее и меньше зависят от изменений в верстке страницы.

Другое важное замечание: хороший тест проверяет только маленькую, атомарную часть функциональности. Простые тесты, которые проверяют небольшой сценарий, лучше, чем один большой тест, проверяющий сразу много сценариев. Благодаря простым тестам быстрее локализуется место в продукте, где появился баг, а также есть возможность найти одновременно несколько новых багов. Упавший большой автотест укажет только на первую встреченную проблему, т. к. он заканчивает работу при первой же найденной ошибке. В этом их отличие от ручных тестов, в которых, проверяя функциональность продукта по тест-кейсу, можно гибко обойти встречающиеся проблемы и пройти тест-кейс до конца, найдя все баги.

Рассмотрим следующий пример: у нас есть форма регистрации, в которой есть обязательные и необязательные поля для заполнения. Нужно проверить, что можно успешно зарегистрироваться на сайте.

Сценарий плохого автотеста:

1

- Открыть страницу с формой
- Заполнить все поля
- Нажать кнопку «Регистрация»
- Проверить, что есть сообщение об успешной регистрации

Лучше разбить предыдущий тест на набор более простых автотестов:

2

- Открыть страницу с формой
- Заполнить только обязательные поля
- Нажать кнопку «Регистрация»
- Проверить, что есть сообщение об успешной регистрации

3

- Открыть страницу с формой
- Заполнить все обязательные поля
- Заполнить все необязательные поля
- Нажать кнопку «Регистрация»
- Проверить, что есть сообщение об успешной регистрации

4

- Открыть страницу с формой
- Заполнить только необязательные поля
- Проверить, что кнопка «Регистрация» неактивна

Попробуем реализовать один из автотестов из предыдущего шага.

Дана страница (https://suninjuly.github.io/registration1.html) с формой регистрации. Проверьте, что можно зарегистрироваться на сайте, заполнив только обязательные поля, отмеченные символом «*»: First name, last name,

e-mail. Текст для полей может быть любым. Успешность регистрации проверяется сравнением ожидаемого текста «Congratulations! You have successfully registered!» с текстом на странице, которая открывается после регистрации. Для сравнения воспользуемся стандартной конструкцией assert из языка Python.

Ниже дан шаблон кода, который нужно использовать для своего теста. Не забывайте, что селекторы должны быть уникальными.

```
from selenium import webdriver
import time
try:
    link = "http://suninjuly.github.io/registration1.html"
    browser = webdriver.Chrome()
    browser.get(link)
    # Ваш код, который заполняет обязательные поля
    ...
    # Отправляем заполненную форму
    button = browser.find_element_by_css_selector("button.btn")
    button.click()
    # Проверяем, что смогли зарегистрироваться
    # ждем загрузки страницы
    time.sleep(1)
    # находим элемент, содержащий текст
    welcome_text_elt = browser.find_element_by_tag_name("h1")
    # записываем в переменную welcome_text текст из элемента welcome_text_elt
    welcome_text = welcome_text_elt.text
    # с помощью assert проверяем, что ожидаемый текст совпадает с текстом на странице сайта
    assert "Congratulations! You have successfully registered!" == welcome_text
finally:
    # ожидание, чтобы визуально оценить результаты прохождения скрипта
    time.sleep(10)
    # закрываем браузер после всех манипуляций
    browser.quit()
```

Использование конструкции assert

Если результат проверки «Поздравляем! Вы успешно зарегистрировались!» == welcome_text вернет значение *False*, то далее выполнится код *assert False*. Он выведет исключение *AssertionError* и номер строки, в которой произошла ошибка. Если код написан правильно и работал ранее, то такой результат равносилен тому, что автотест обнаружил баг в

тестируемом веб-приложении. Если результат проверки вернет *True*, то выполнится выражение *assert True*. В этом случае код завершится без ошибок – тест прошел успешно.

Обратите внимание. В этом примере использован метод *time.sleep(1)*, чтобы дождаться загрузки следующей страницы, прежде чем выполнять проверки. Без использования данного метода WebDriver может перейти к поиску тега *h1* слишком рано, когда новая страница еще не загрузилась. В таком случае в терминале будет следующая ошибка:

NoSuchElementException... Unable to locate element: {"method":"tag name","selector":"h1"}

Метод *time.sleep(1)* говорит Python подождать 1 секунду, прежде чем выполнять следующую строчку кода. Если все равно видна ошибка, просто увеличьте количество секунд ожидания.

Практическое задание:

1. Получить у преподавателя программный продукт для тестирования.
2. Протестировать программный продукт используя библиотеку Selenium.
3. Зафиксировать полученные результаты тестирования.
4. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

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

Контрольные вопросы:

1. Что такое Selenium?
2. Какие стратегии позволяют производить поиск по атрибутам элементов?
3. Факторы, влияющие на стабильность работы автоматизированных тестов?
4. Какие команды в Selenium позволяют искать элементы?
5. Что такое Page Object Model?
6. Как работает XPath-селектор?
7. В чем разница результатов команд find_element и find_elements?
8. В каком случае будет выдаваться ошибка No Such Element Exception?
9. Какой универсальный метод поиска в Selenium?
10. В чем заключается уникальность селектора?
11. Для чего используется метод time.sleep(1)?
12. В каких случаях метод time.sleep(1) нежелательно использовать?

Лабораторная работа № 6

Тестирование API

Цель: разработать автоматизированные тесты с использованием инструмента Postman.

План занятия:

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

Теоретические сведения

API (Application Programming Interface или интерфейс программирования приложений) – это совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой. Это позволяет разработчикам расширять функциональность своего продукта и связывать его с другими.

Интерфейс представляет собой промежуточный слой между двумя приложениями. Он позволяет двум программам обмениваться информацией и выполнять функции, не раскрывая своего внутреннего API. Скрытие части функций называется инкапсуляцией.

Есть три метода взаимодействия с API:

1. Процесс, который может выполнять программа при помощи этого интерфейса.
2. Данные, которые нужно передать интерфейсу для выполнения им функции.
3. Данные, которые программа получит на выходе после работы с API.

Разработчик имеет полную свободу в выстраивании функций API. Например, отдельный набор функций может определять возможность регистрироваться и авторизоваться в программе.

API бывают публичные и частные. Первые предназначены для совместного использования с внешним миром, например VK API. Сторонние разработчики могут создавать приложения, чтобы воспользоваться возможностями этих интерфейсов. Вторые – это внутренние приложения, разработанные для определенной аудитории или пользовательской базы. Они часто используются на предприятиях и внутри компаний. Для работы с таким API нужно получить доступ.

Использование API.

Разработчикам программный интерфейс позволяет:

1) упростить и ускорить выпуск новых продуктов, т. к. можно использовать уже готовые API для стандартных функций;

2) сделать разработку более безопасной, выведя ряд функций в отдельное приложение, где они будут скрыты;

3) упростить настройку связей между разными сервисами и программами и не сотрудничать для разработки своего продукта с создателями различных приложений;

4) сэкономить деньги, т. к. не нужно разрабатывать все программные решения с нуля.

До появления Windows и других графических операционных систем программистам для создания окон на экране компьютера приходилось писать тысячи строк кода. Когда же Microsoft предоставила разработчикам API Windows, на создание окон стало уходить всего несколько минут работы.

Бизнесу API нужны, чтобы:

1) проводить транзакции;

2) интегрировать потоки данных с клиентами и партнерскими системами;

3) повысить безопасность автоматизированных процессов;

4) развивать собственные приложения;

5) внедрять инновации, например, при работе с клиентами.

В 1990-е годы организация, которая хотела запустить систему управления взаимоотношениями с клиентами (CRM), была вынуждена вкладывать огромные средства в программное обеспечение, оборудование и специалистов. Теперь компании используют облачные службы вроде Salesforce. Доступ на уровне API к функциям Salesforce позволяет бизнесу включить ключевые элементы функциональности CRM-системы – например, возможность просматривать историю клиента.

Правительствам API позволяют:

1) обмениваться данными между ведомствами;

2) взаимодействовать с гражданами, получать обратную связь.

Тестирование API является важной частью разработки программного обеспечения, но при выполнении вручную оно может отнимать много времени и включать в себя много повторяющихся задач. Postman является одним из наиболее широко используемых инструментов для тестирования API. Однако многие пользователи часто не используют его возможности автоматизации в полной мере, что приводит к снижению эффективности процесса тестирования API.

Принято проводить ручное тестирование только для API, которые все еще находятся в разработке, или в определенных ситуациях, когда автоматизация может быть не лучшим вариантом, например при исследовательском, ad-hoc, юзабилити и тестировании пограничных случаев. Для стабильных API, уже находящихся в продакшене, полагаются на автоматизированное тестирование с использованием заранее написанных тестовых сценариев для получения эффективных и точных результатов.

Рассмотрим пример тестирования API. Для этого будем использовать коллекцию Postman Collection, представленную ниже. API, применяемые в этой

демонстрации, взяты с сайта gorest.co.in, который предоставляет бесплатный фиктивный сервис REST API, предназначенный для тестирования (рисунок 29).

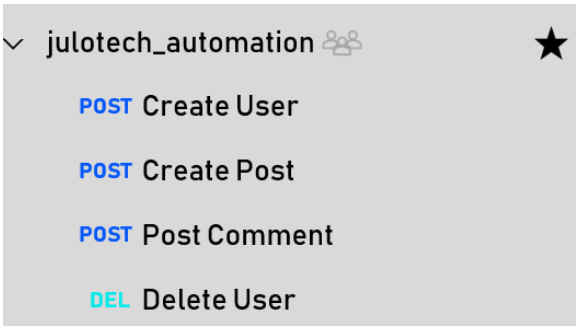

Рисунок 29 – Окно с запросами

Названия запросов понятны и не требуют пояснений. Однако для контекста предположим, что эти API относятся к типичной системе онлайн-форума, которая включает функциональность для создания пользователей, сообщений, комментариев и удаления пользователей.

Интеграционное тестирование часто предполагает использование данных из ответа одного API в качестве параметра другого API. Например, чтобы протестировать API 'Create Post' и 'Delete User', необходимо добавить в URL ID пользователя, полученный из ответа API 'Create User' (рисунки 30–32).

Рисунок 30 – Получение user ID из успешного ответа API 'Create User'

Рисунок 31 – Размещение user ID в URL API 'Create Post'

Рисунок 32 – Ввод user ID в URL-адрес API 'Delete User'

'

Аналогично URL API 'Post Comment' требует Post ID, полученный из ответа API 'Create Post'. Эти шаги могут быть утомительными и отнимать много времени, особенно при тестировании реальной системы, которая обычно включает в себя больше четырех API.

Также необходимо проводить негативные тесты на каждом API в дополнение к интеграционному тестированию. Давайте рассмотрим несколько примеров негативных тест-кейсов для API 'Create User' (рисунки 33 и 34).

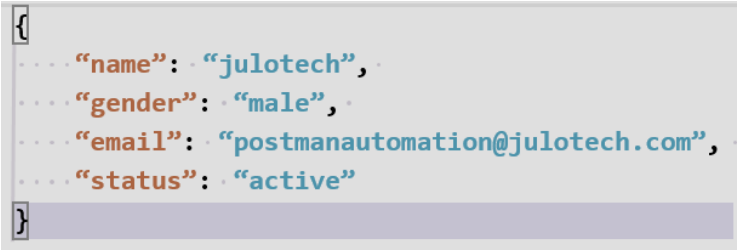

Рисунок 33 – Тело запроса API 'Create User' (JSON)

Number	Test Case
1	Check if “name” is empty
2	Check if “gender” is other than “male” or “female”
3	Check if “gender” is empty
4	Check if “email” is in invalid format
5	Check if “email” is already taken
6	Check if “email” is empty
7	Check if “status” is other than “active” or “inactive”
8	Check if “status” is empty

Рисунок 34 – Примеры негативных тест-кейсов для API 'Create User'

Начнем автоматизировать API. Разделим процесс на две части: интеграция и управление данными.

Автоматизация – интеграционный тест. Использование переменных

Можно упростить процесс, сохранив идентификатор пользователя из ответа API 'Create User' в переменной и затем используя его в URL API 'Create Post' и 'Delete User', вместо того, чтобы вручную копировать и вставлять его несколько раз (рисунок 35).

Эту задачу можно решить, просто включив небольшой фрагмент кода на JavaScript в раздел Tests запроса. После отправки запроса Postman автоматически выполнит код в разделе Tests.

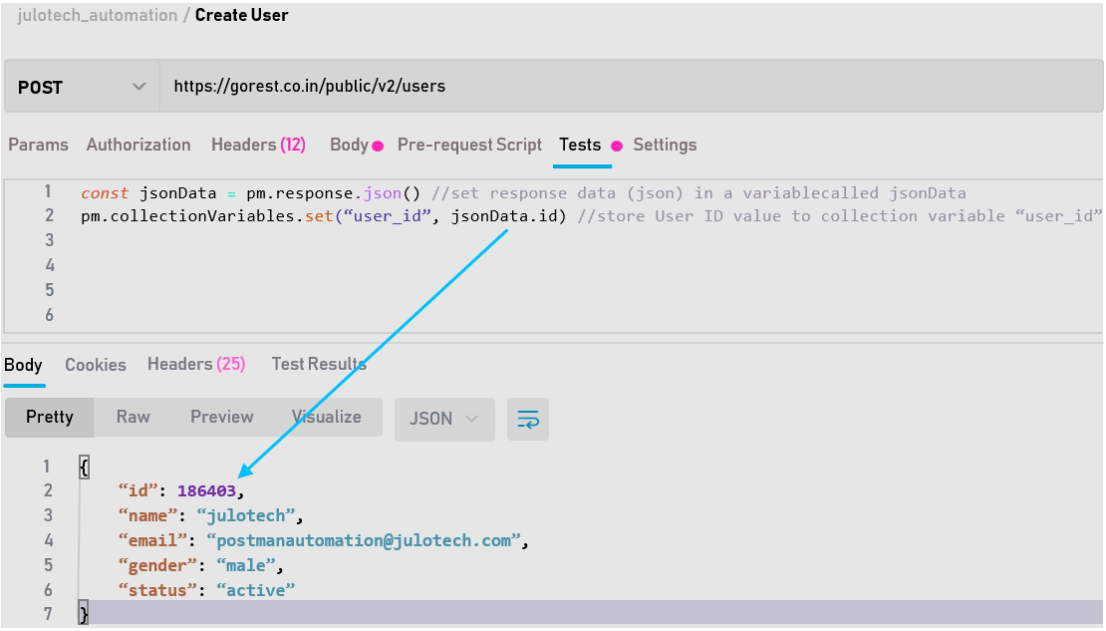

Рисунок 35 – Сохранение User ID в переменной коллекции “user_id”

Теперь, когда ID пользователя сохранен в переменной коллекции с именем “user_id”, давайте рассмотрим переменные коллекции. Переменная “user_id” теперь должна содержать значение User ID, полученное из ответа (рисунок 36).

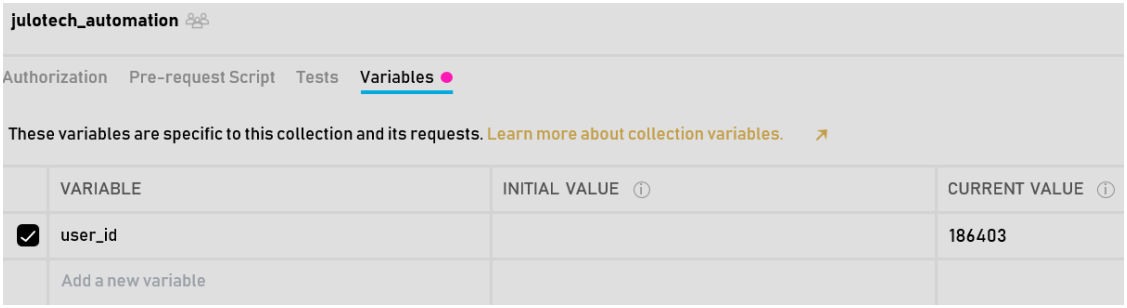

Рисунок 36 – Переменные коллекции

Наконец, включите переменную в URL-адреса API 'Create Post' и 'Delete User' (рисунки 37 и 38).

Рисунок 37 – ‘Create Post’ API URL

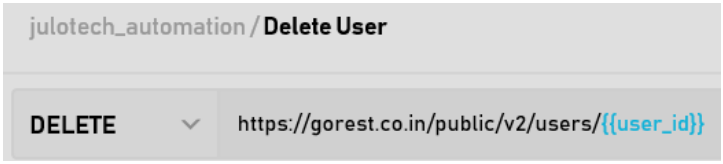

Рисунок 38 – ‘Delete User’ API URL

Поскольку URL API 'Post Comment' требует Post ID, полученный из ответа API 'Create Post', давайте сделаем то же самое (рисунки 39 и 40).

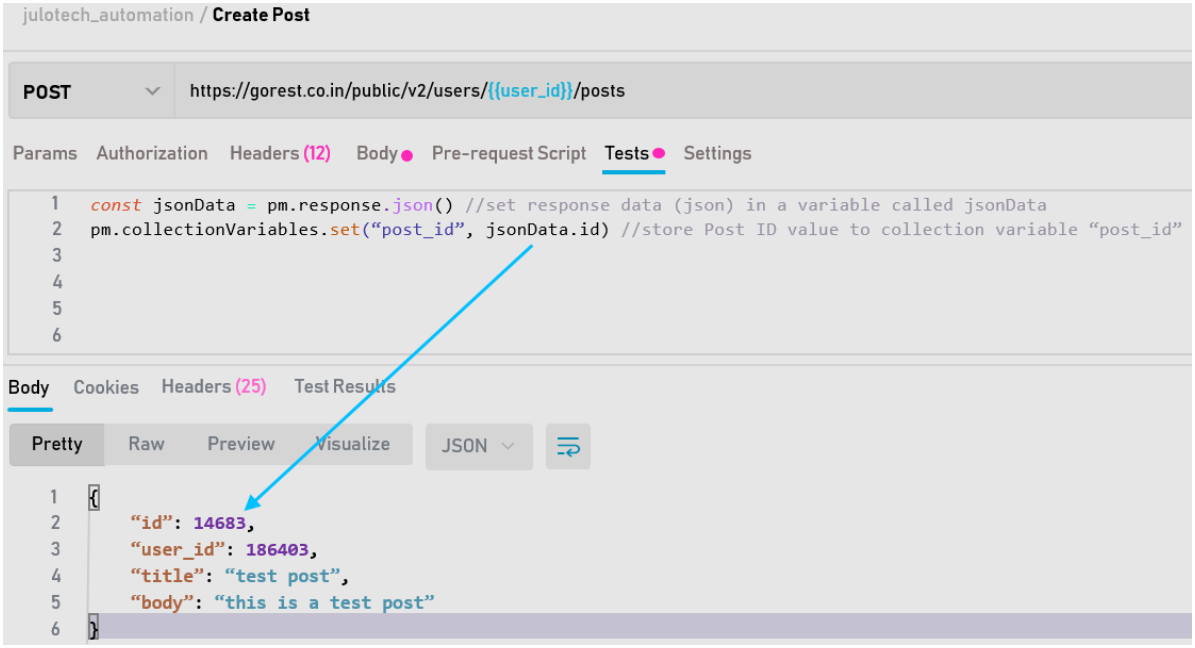

Рисунок 39 – Сохранение Post ID в переменной коллекции “post_id”

Рисунок 40 – URL API 'Post Comment'

По завершении этого шага необходимость в ручном копировании и вставке данных между ответами API и URL будет устранена.

Создание утверждений (Assertions).

Утверждения являются важным компонентом автоматизации, поскольку они позволяют определить успех или неудачу теста.

Аналогичным образом утверждения могут быть включены в раздел Tests запроса. Поскольку типов утверждений существует большое количество, сосредоточимся на процессе добавления утверждений для код-статуса ответа и тела ответа (рисунки 41 и 42).

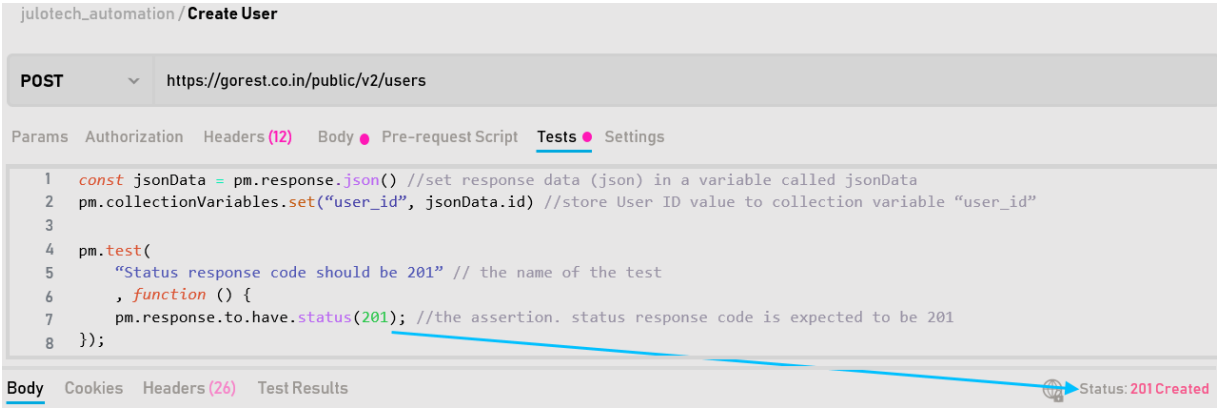

Рисунок 41 – Утверждение кода ответа состояния

Рисунок 42 – Утверждение тела ответа

Давайте сделаем проверку: отправим запрос и посмотрим на раздел «Результаты тестирования / Test Results» (рисунок 43).

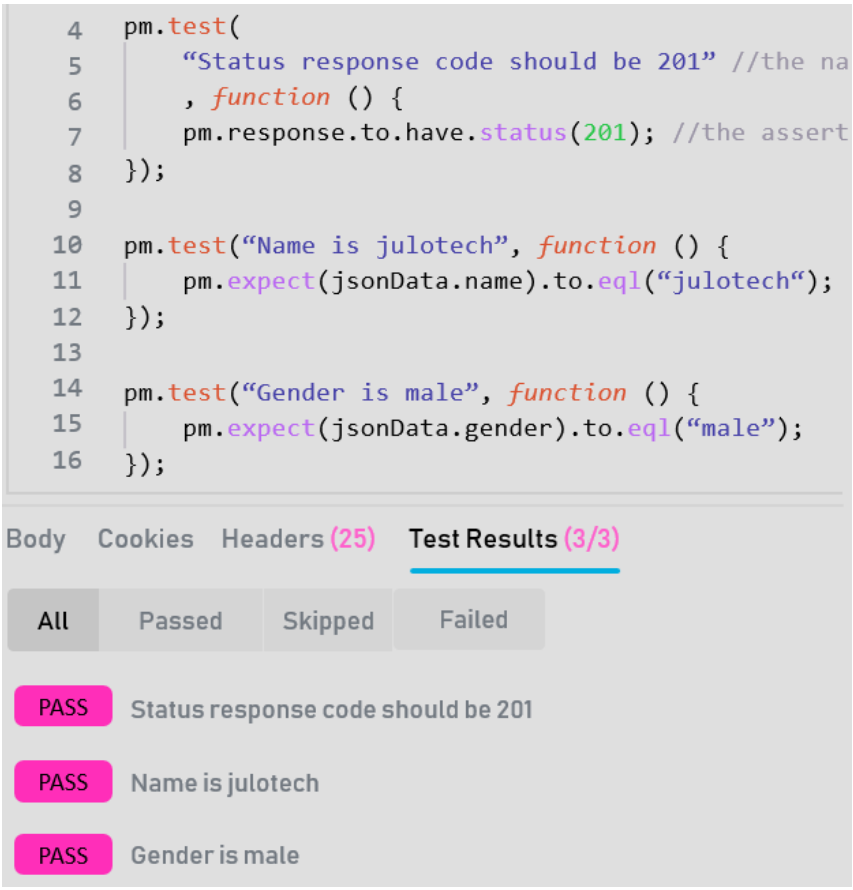

Рисунок 43 – Просмотр результатов тестирования

Теперь необходимо применить тот же процесс к остальным API.

Использование Postman Collection Runner

Postman Collection Runner автоматизирует процесс, запуская все API в коллекции. Помните, что он запускает их последовательно, поэтому перед запуском *runner*-а убедитесь, что порядок API в коллекции правильный.

1. Нажмите на три точки рядом с названием коллекции и выберите Run collection (рисунок 44).

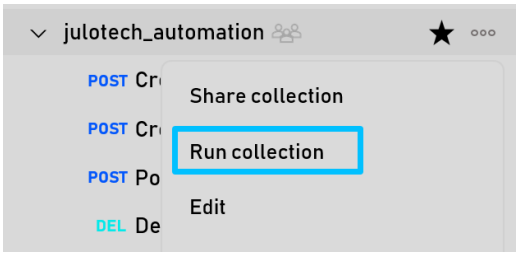

Рисунок 44 – Выбор коллекции

2. Нажмите кнопку «Run <имя коллекции>» (рисунок 45).

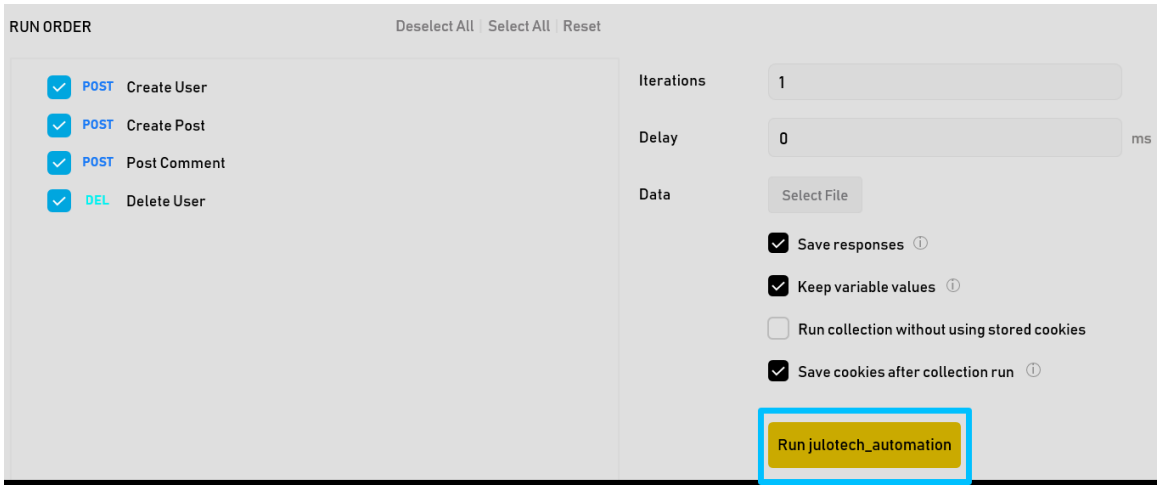

Рисунок 45 – Выбор коллекции с именем

3. Результат автоматизации теста представлен на рисунке 46.

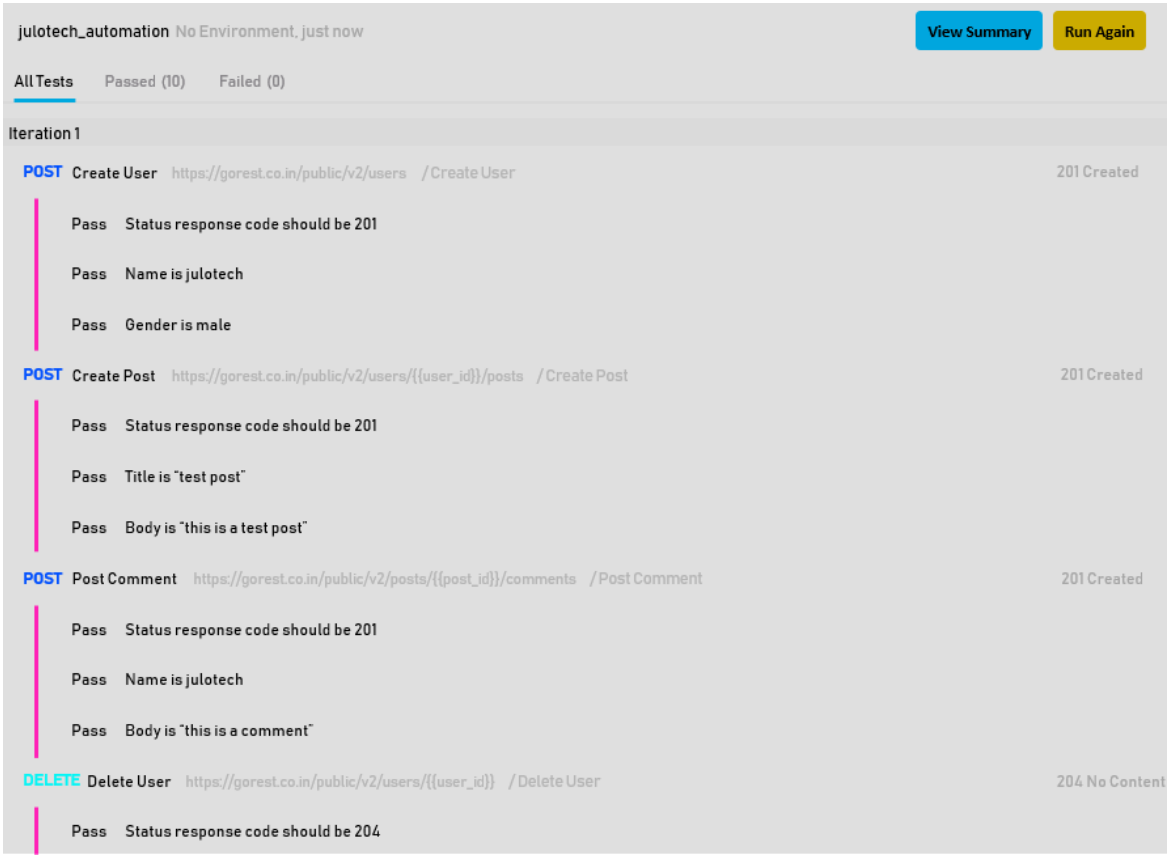

Рисунок 46 – Автоматизация теста

Теперь интеграционный тест автоматизирован, что позволяет просто и без особых усилий выполнять его в любое время.

Автоматизация – тест, основанный на данных. Настройка запросов

Цель этого процесса – создать наборы тестовых данных в CSV-файле, сохранить их в переменной коллекции, а затем отправить запрос определенное количество раз, исходя из количества строк (как тест-кейсов) в файле.

1. Создайте CSV-файл и введите необходимые тестовые данные (рисунок 47).

Number	Test Case	name	gender	email	status
1	Check if "name" is empty		male	postmanautomation@julotech.com	active
2	Check if "gender" is other than "male" or "female"	julotech	asd	postmanautomation@julotech.com	active
3	Check if "gender" is empty	julotech		postmanautomation@julotech.com	active
4	Check if "email" is in invalid format	julotech	male	postmanautomationjulotech.com	active
5	Check if "email" is already taken	julotech	female	postmanautomation123@julotech.com	active
6	Check if "email" is empty	julotech	male		active
7	Check if "status" is other than "active" or "inactive"	julotech	male	postmanautomation@julotech.com	asd
8	Check if "status" is empty	julotech	female	postmanautomation@julotech.com	

Рисунок 47 – Создание CSV-файла

2. Назначьте отдельную переменную для значения каждого атрибута (рисунок 48).

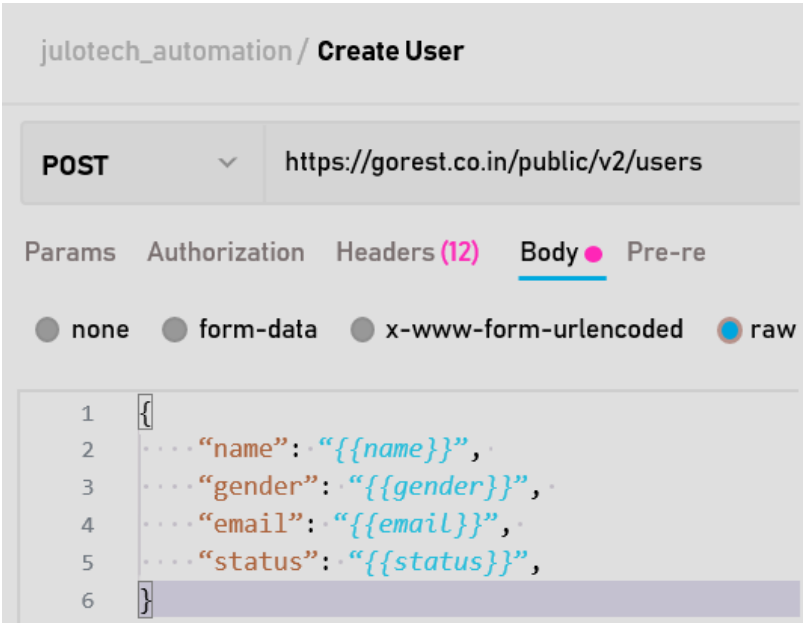

Рисунок 48 – Назначение переменных для атрибутов

3. В разделе Pre-request Script добавьте код, чтобы загрузить данные теста из CSV-файла и сохранить их в переменных коллекции. Обратите внимание, что Postman автоматически выполнит этот код перед отправкой запроса, в отличие от раздела «Тесты» (рисунок 49).

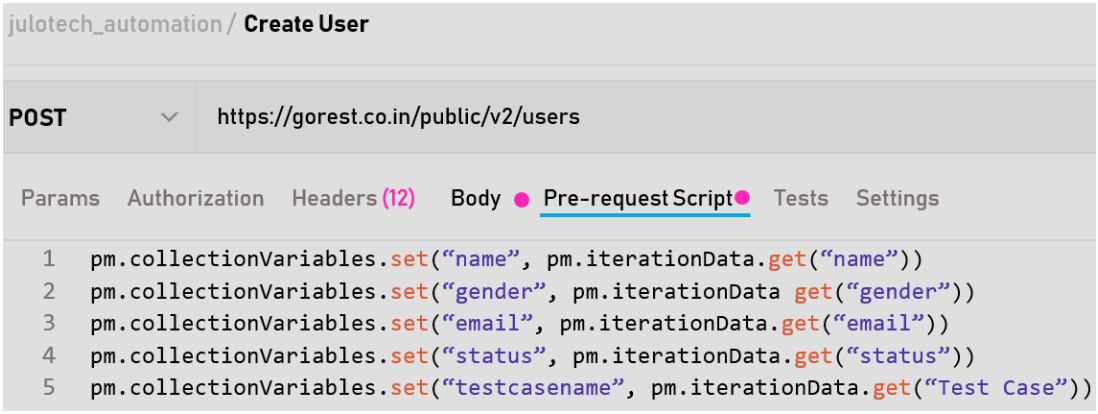

Рисунок 49 – Добавление кода в раздел

Создание утверждения

На рисунке показан результат тест-кейса «проверить, пусто ли имя» (рисунок 50).

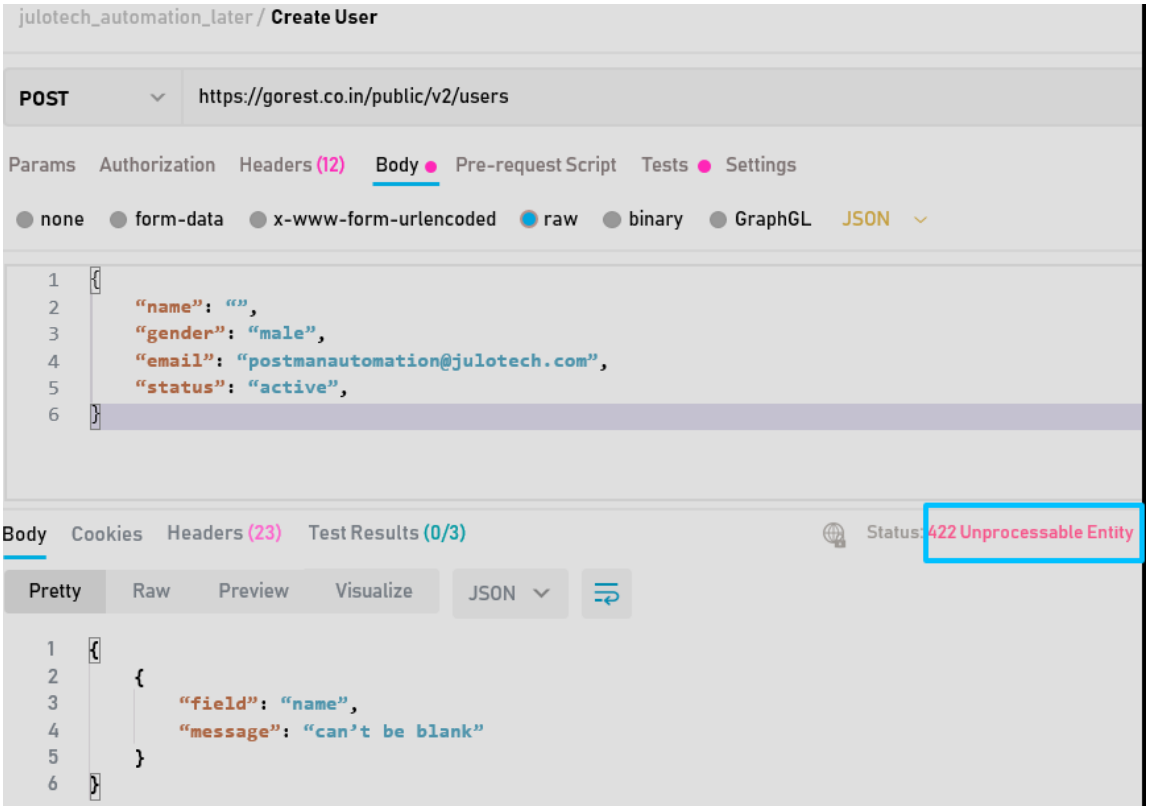

Рисунок 50 – Результат тест-кейса

Поскольку все тест-кейсы будут возвращать ответ со статус-кодом 422, будет целесообразным определить это значение в качестве ожидаемого ответа. Кроме того, нам понадобится еще один «Тест» для отображения описания каждого тест-кейса. Для этой информационной цели утверждение не требуется (рисунок 51).

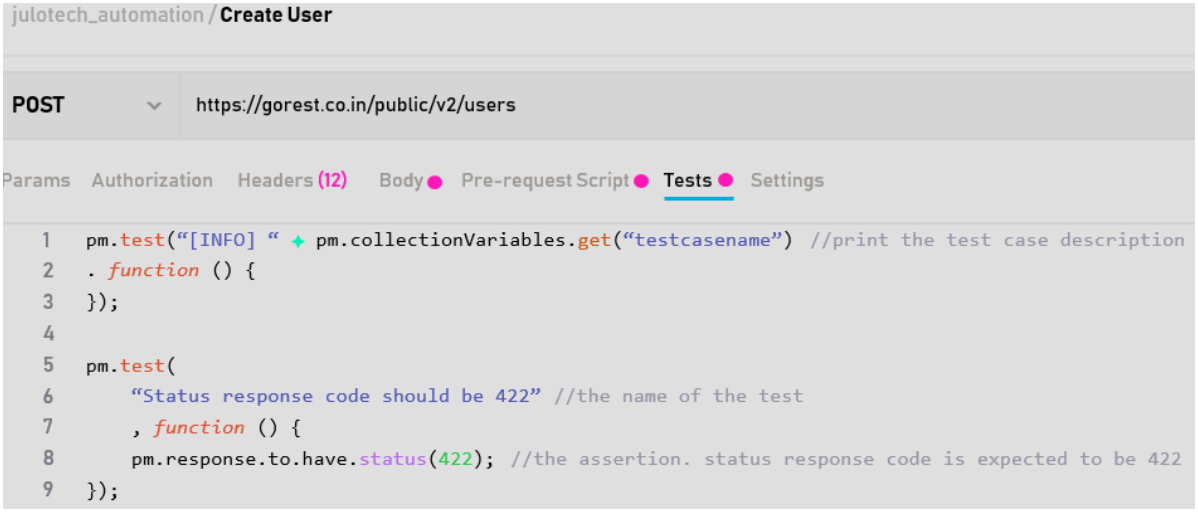

Рисунок 51 – Статус-код 422

Загрузка CSV-файла и запуск коллекции

1. Перед запуском коллекции загрузите CSV-файл, содержащий тестовые данные. Postman автоматически определит количество итераций (рисунок 52).

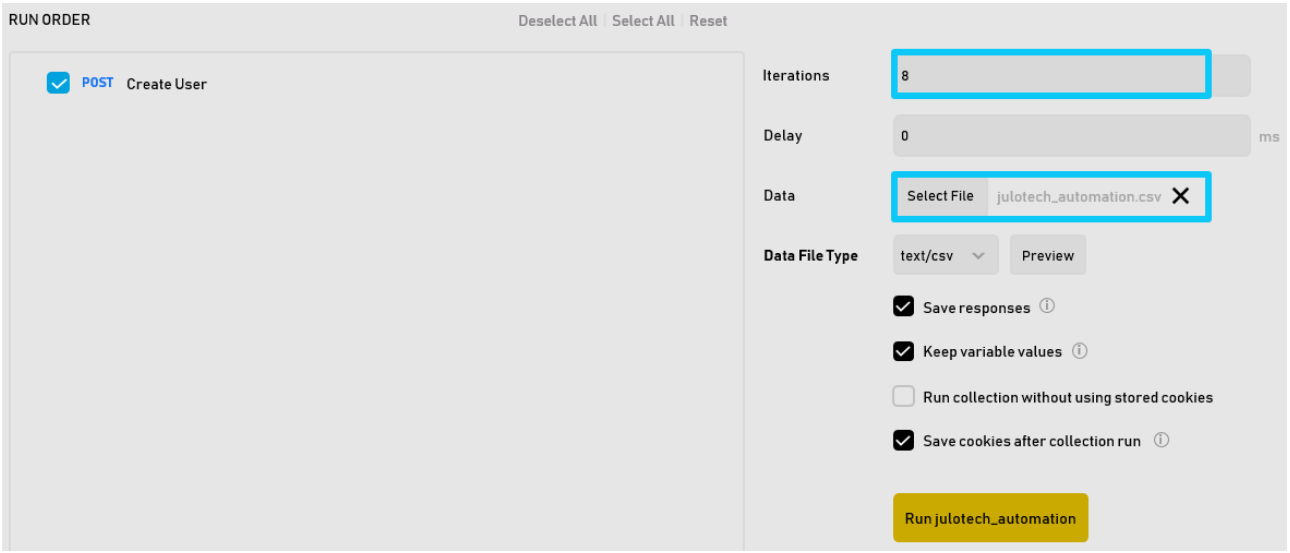

Рисунок 52 – Загрузка CSV-файла

2. Нажмите на кнопку «Запустить/Run <имя коллекции>» (рисунки 53 и 54).

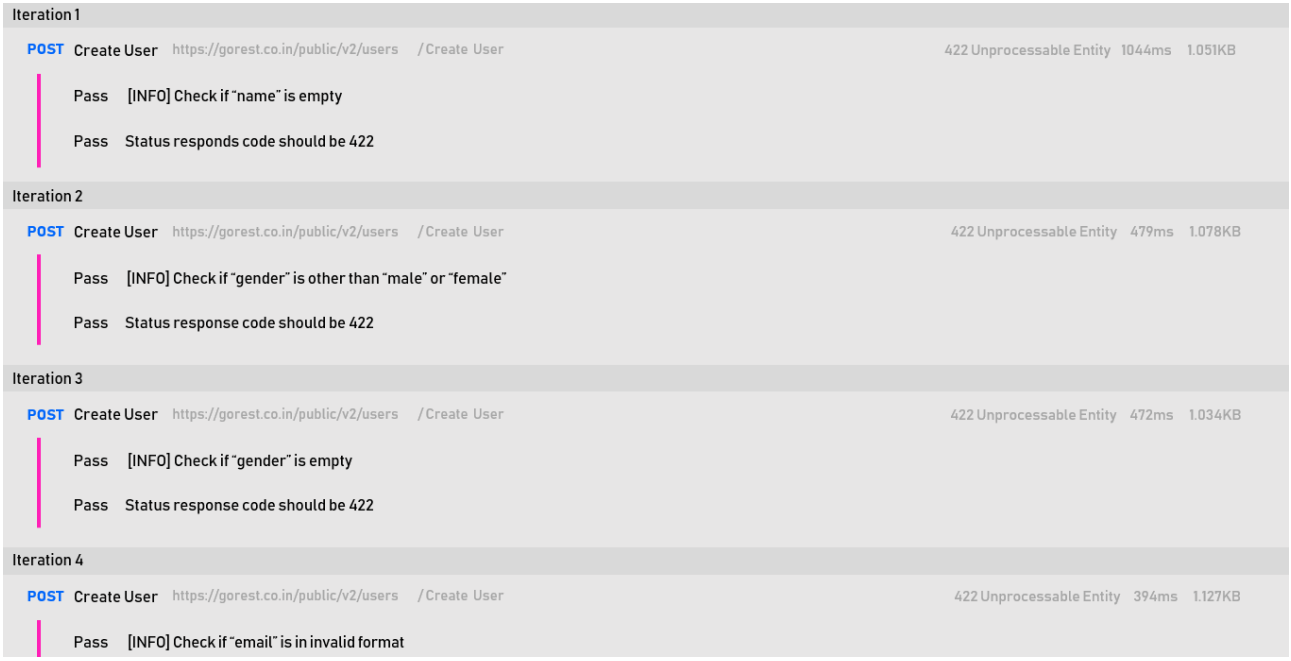

Рисунок 53 – Результаты выполнения тест-кейсов с 1 по 4

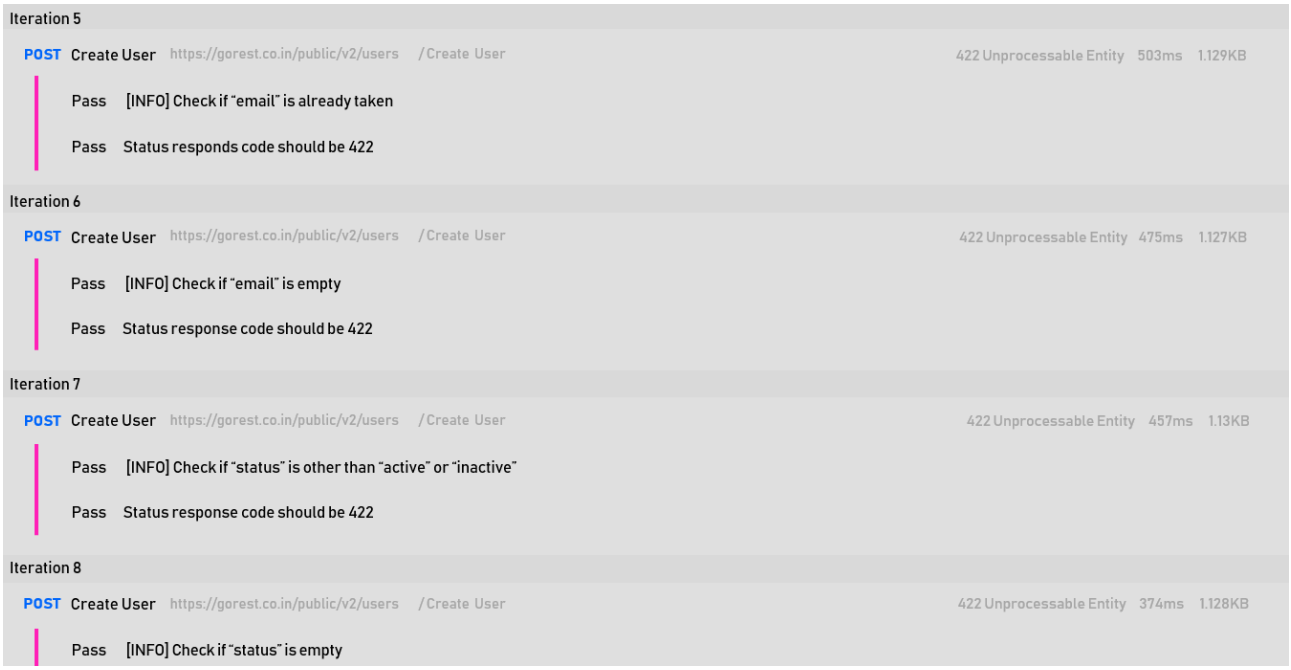

Рисунок 54 – Результаты выполнения тест-кейсов с 5 по 8

Из результатов видно, что все утверждения прошли. Можно для проверки углубиться в детали каждого утверждения, как в приведенном ниже примере проверки, когда параметр name пуст (рисунок 55).

Рисунок 55 – Тест-кейс 1

В результате видно, что все отправленные запросы соответствуют тестовым данным из CSV-файла.

Автоматизированный Data Driven тест завершен.

Практическое задание:

1. Получить у преподавателя программный продукт для тестирования.
2. Протестировать API программного продукта.
3. Зафиксировать полученные результаты тестирования.
4. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

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

Контрольные вопросы:

1. Что такое API?
2. Как работает API?
3. Какие существуют три метода взаимодействия с API?
4. Назовите существующие виды API.
5. Для чего используют API?
6. Что такое Postman?
7. Что позволяет разработчикам программный интерфейс?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Савин Р. Тестирование dot com / Р. Савин. – Екатеринбург : Ridero, 2017. – 312 с.

2. Назина, О. Что такое тестирование : курс молодого бойца / О. Назина. – СПб. : БХВ-Петербург, 2024. – 592 с.

3. Аниче, М. Эффективное тестирование программного обеспечения : руководство разработчика / М. Аниче ; [пер. с англ. А. Н. Киселёва]. – М. : ДМК Пресс, 2023. – 370 с.

4. ISTQB Стандартный глоссарий терминов, используемых в тестировании программного обеспечения. – 2014. – 73 с.

5. Тестирование программного обеспечения : учеб. пособие / С. С. Куликов [и др.]. – Минск : БГУИР, 2019. – 277 с.

Св. план 2025, поз. 21

Учебное издание

Василькова Анастасия Николаевна
Воробей Анастасия Владимировна
Медведев Олег Сергеевич
Прудник Александр Михайлович

ТЕХНОЛОГИИ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ПОСОБИЕ

Редактор *Ю. В. Граховская*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 16.09.2025. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 5,23. Уч. изд. л. 5,3. Тираж 60 экз. Заказ 35.

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