# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc105494147)

[ВВЕДЕНИЕ 5](#_Toc105494148)

[1 Обзор способов тестирования программного обеспечения 7](#_Toc105494149)

[1.1 Обзор подходов к тестированию программного обеспечения 9](#_Toc105494150)

[1.2 Схема работы специалистов по тестированию 12](#_Toc105494151)

[1.3 Обзор подходов к автоматизации тестирования ПО 15](#_Toc105494152)

[2 Разработка структуры фреймворка с гибридным подходом 19](#_Toc105494153)

[2.1 Архитектура модулей гибридного фреймворка 21](#_Toc105494154)

[2.2 Шаблоны для унификации формирования тестовых сценариев 26](#_Toc105494155)

[2.3 Алгоритмы работы в модулях 27](#_Toc105494156)

[2.4 Реализация модулей 32](#_Toc105494157)

[2.5 Пример применения типовой модели автоматизированных тестов 38](#_Toc105494158)

[3 Оценка юзабилити разработанного фреймворка 45](#_Toc105494159)

[3.1 Метод экспертных оценок 45](#_Toc105494160)

[3.2 Оценка юзабилити разработанного фреймворка 46](#_Toc105494161)

[3.3 Проверка правильности нормализации ранжировок 49](#_Toc105494162)

[3.4 Вычисление коэффициентов важности критериев 50](#_Toc105494163)

[3.5 Балльная оценка качества разработанного фреймворка 51](#_Toc105494164)

[3.6 Расчёт оценки качества разработанной системы 52](#_Toc105494165)

[ЗАКЛЮЧЕНИЕ 53](#_Toc105494166)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 54](#_Toc105494167)

[ПРИЛОЖЕНИЕ А 55](#_Toc105494168)

[ПРИЛОЖЕНИЕ Б 58](#_Toc105494169)

[ПРИЛОЖЕНИЕ В 60](#_Toc105494170)

[ПРИЛОЖЕНИЕ Г 64](#_Toc105494171)

# ВВЕДЕНИЕ

В современном мире внедрение информационных технологий становится неотъемлемым требованием в различных областях человеческой деятельности.

Крупным IT компаниям приходится постоянно решать новые задачи, гибко подстраиваясь под требования рынка. Постоянно растёт потребность в разработке нового программного обеспечения. Решения находят быстро, но зачастую они не всегда оказываются наиболее эффективными.

Каждая новая программа (система), проходит несколько жизненно важных этапов разработки. Одним из этапов является тестирование программного обеспечения. Задачи организации тестирования программного обеспечения каждая компания решает по-своему, но проблема остаётся на сегодняшний день открытой.

Для проведения тестирования ПО требуется постоянно растущий штат специалистов и серьезные временные затраты, что оборачивается для компании большими убытками.

Проблема является наиболее актуальной для крупных компаний, в которых тестированию подлежит большой объем создаваемого программного обеспечения.

Цель исследования – сделать процесс тестирования программного обеспечения гибким, не ресурсоемким и удобным.

Задачи выпускной квалификационной работы:

* проанализировать существующие подходы к автоматизации тестирования в крупных компаниях;
* предложить подход для решения проблем автоматизации тестирования;
* разработать архитектуру модулей гибридного фреймворка;
* разработать шаблоны для унификации формирования тестовых сценариев;
* разработать доступного удобный и доступный для специалистов по ручному тестированию конструктор авто-тестов;
* представить типовую модель автоматизации тестирования на контрольном примере.

Выпускная квалификационная работа состоит из трех глав.

В главе 1 рассмотрен цикл создания программного обеспечения, проанализирован сам процесс тестирования, его разновидности, способы эксплуатации и возникающие проблемы. Так же были рассмотрены подходы к проведению тестирования в крупных компаниях, способы внедрения автоматических тестов, возникающие проблемы и предложен способ их решения.

В главе 2 представлена разработанная архитектура будущего приложения для автоматизации тестирования, рассмотрены все модули, задействованные в целевом решении, представлены основные алгоритмы работы с разработанным программным обеспечением и внутренний алгоритм работы центрального модуля, приведены основные используемые концепции конечной реализации на языке Java. Приведён контрольный пример.

В главе 3 представлены расчёты юзабилити полученного фреймворка методом экспертных оценок, для оценки приложения была отобрана группа сотрудников, проведён анализ нового решения, проставлены оценки по каждому критерию и подведён общий итог.

## Обзор способов тестирования программного обеспечения

Современный цикл разработки программного обеспечения совмещает в себе несколько этапов, представленных на рисунке 1.

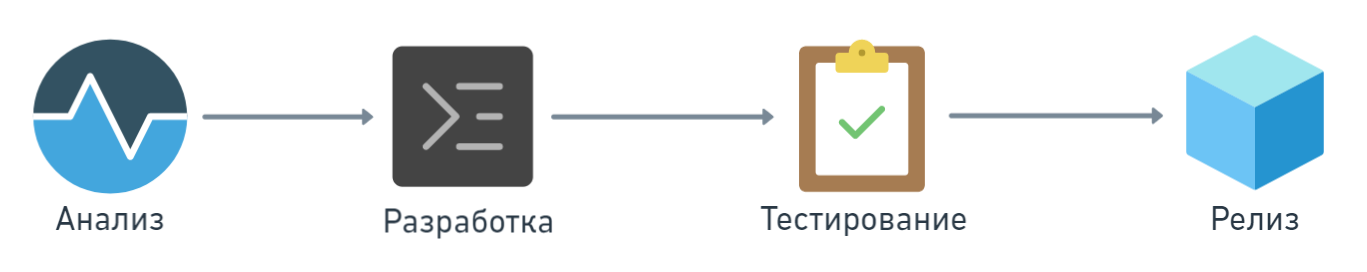


Рисунок 1 - Современный цикл разработки ПО

Анализ — обсуждение задач с заказчиком, уточнение требований к конечному продукту, трансляция заявленных требований разработчикам, разработка технической документации и постановка задачи [1].

Разработка — это большой и непрерывный этап жизненного цикла, выполняется реализация технического задания, бизнес требований, планировка архитектуры будущей информационной системы. Происходит корректировка требований, обсуждение новой функциональности, доработка технического задания [4].

Тестирование — постоянный процесс тестирования приложения, которое находится в разработке. На этом этапе проверяется не только его соответствие требованиям, но и пропускная способность, отказоустойчивость, готовность к непредвиденным ситуациям, к различному поведению пользователей внутри и снаружи будущей системы [1] [3] [5] [6].

Релиз — поставка приложения, которое успешно прошло этап тестирования, заказчику для уточнения требований, возможных доработок и непосредственной эксплуатации [4].



Рисунок 2 - Схема взаимодействия разработчика, тестировщика и заказчика

На рисунке 2 представлена схема взаимодействия разработчика, тестировщика и заказчика. Разработчик постоянно реализует новый функционал для приложения и исправляет выявленные ошибки. Тестировщик занимается проверкой работы приложения и передаёт его заказчику. Заказчик в свою очередь получает приложение и предлагает идеи по его улучшению, исправлению не выявленных при тестировании ошибок и т.д.

### Обзор подходов к тестированию программного обеспечения

Проблема тестирования прежде всего связана с огромными объёмами тестов. Среднестатистическое приложение крупной компании скрывает за собой многогранную и сложную архитектуру программного обеспечения, состоящую из сотни, а зачастую из тысяч небольших приложений, работающих как единое целое [4] [5].

Такой объём необходимо непрерывно тестировать, чтобы обеспечить стабильность и надёжность приложений, потому и разрабатываются подходы для решения подобных задач. С их помощью получится обеспечить тестами приложение любых размеров.

Тестирование программного обеспечения делится на множество подтипов, представленных на рисунке 3.

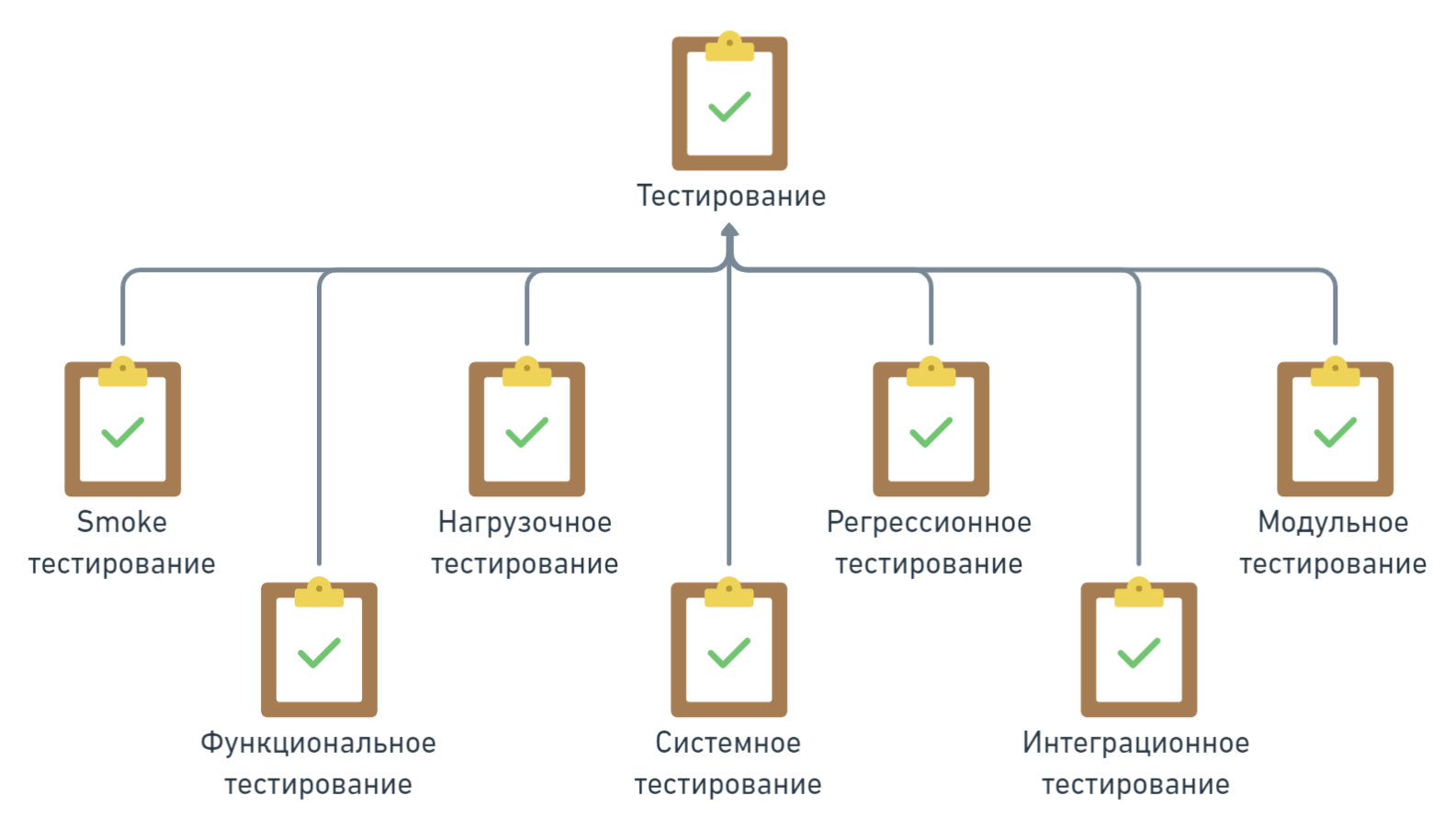


Рисунок 3 - Виды тестирования

Модульное тестирование. Такие тесты пишут разработчики для проработки небольших участков кода. Они позволяют избежать явных логических ошибок. Модульные тесты обеспечивают покрытием около 80% всего кода, они обязательно включают в себя все методы и логические ветвления [1] [2] [3].

Функциональное тестирование обеспечивает проверку заявленного функционала. Проверяется работа приложения согласно написанной технической документации [5].

Smoke тестирование. Проверка приложения на работоспособность. Проверяется работа приложения независимо от бизнес-требований. Успешное smoke тестирование позволяет приступить к другим типам тестирования [5].

Интеграционное тестирование или A/B тесты. Пишутся тестовые сценарии, по которым будет осуществляться работа с готовым приложением. Позволяет протестировать возможные действия пользователя.  
Интеграционный тест считается проваленным, если не выполнился хотя бы один пункт сценария [1].

Нагрузочное тестирование. Проверяется работа приложения под большой нагрузкой, что позволит увидеть слабые и сильные стороны конечного продукта [1].

Системное тестирование. Данный подтип тестирования проверяет функционал интеграции с другими приложениями или системами [1].

Регрессионное тестирование. Подразумевает под собой множество различных тестов, которые прошли успешно в старой версии приложения. Позволяет легко выявлять ошибки, которые появляются в новых версиях приложения. Количество регрессионных тестов увеличивается с каждым новым релизом [4]. Такой подход к тестированию приложения является приемлемым только на небольших объёмах производства программного обеспечения. Чем больше приложение, тем сложнее становится его тестировать и тем больше времени уходит на его выпуск заказчику [3].

Таким образом, задачи специалиста по тестированию очень широки: ему необходимо изучить документацию, погрузиться в бизнес логику и аспекты разрабатываемого решения, изучить работу всех приложений в целом, написать тестовые сценарии и проверки, которые обеспечат максимальную надёжность будущего продукта.

Также необходимо непрерывно проверять приложение на наличие дефектов следуя написанным планам тестирования и своим домыслам с точки зрения конечного пользователя.

### Схема работы специалистов по тестированию

В современных системах целевым подходом к разработке больших приложений является разбиение его на модули, которые выполняют небольшую часть конечной функциональности. Такие модули принято называть «микросервисы», а архитектура таких приложений называется «микросервисная архитектура» [3] [4] [5]. Распределение работ представлено на рисунке 4.

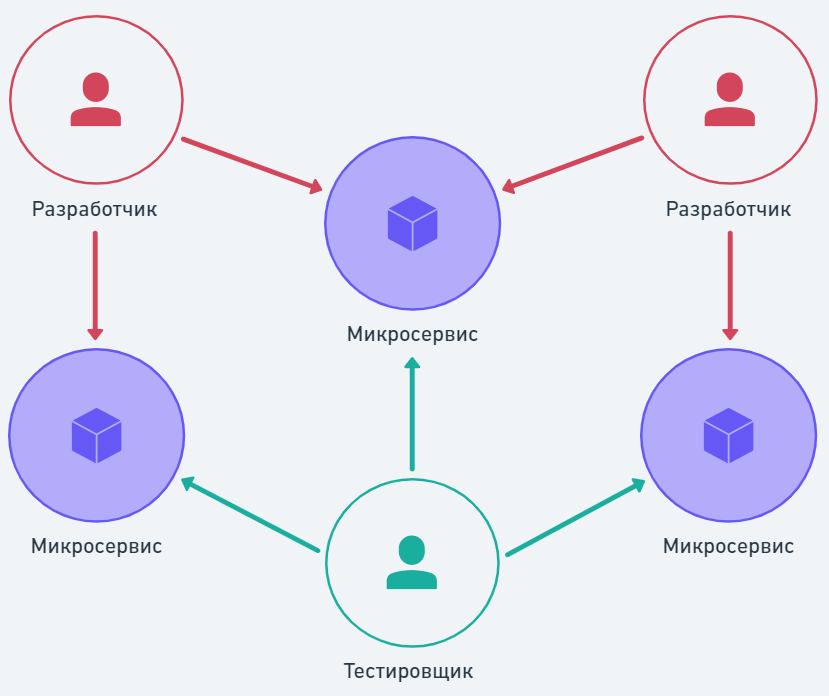


Рисунок 4 - Схема взаимодействия разработчика и тестировщика №1

Достоинства:

* низкие затраты на тестирование;

Недостатки:

* этап тестирования занимает большое количество времени. С каждым новым релизом приложения это время будет расти линейно;
* отсутствует возможность роста команды разработки;
* замедление этапа разработки программного обеспечения, т.к. тестирование не справляется с объёмами текущего приложения.

Решение:

* увеличить команду тестирования. Иллюстрация решения приведена на рисунке 5.

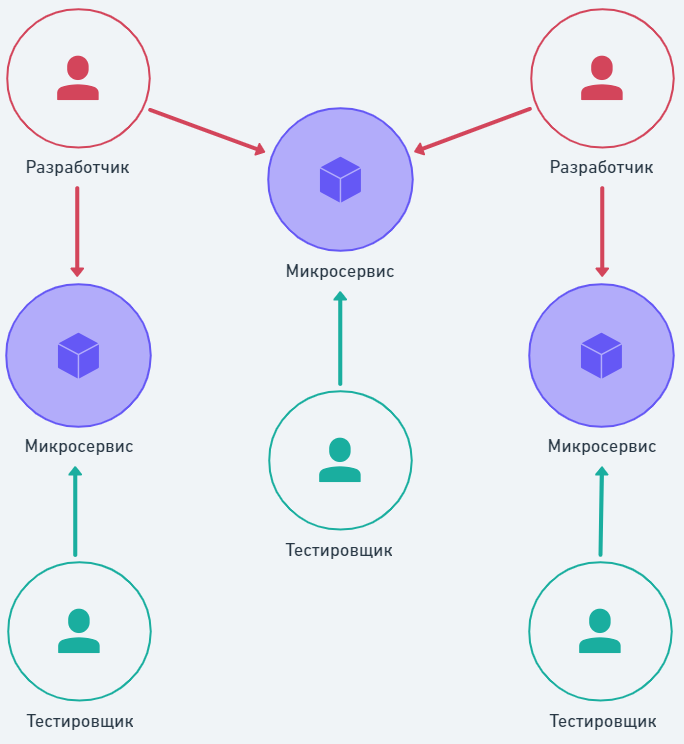


Рисунок 5 - Схема взаимодействия разработчика и тестировщика №2

Достоинства:

* тестирование проходит быстро;
* нагрузка на каждого тестировщика распределена равномерно.

Недостатки:

* рост команды разработки влечёт за собой рост команды тестирования;
* высокие расходы на тестирование;
* после того, как приложение будет полностью готово останется большая команда тестирования.

Решение:

* внедрить автоматизированное тестирование.

### Обзор подходов к автоматизации тестирования ПО

Ручное тестирование – трудоёмкий процесс, который можно автоматизировать. На рынке существует множество решений, которые позволяют работать с браузером, базами данных и API приложений программным путём. Используя автоматизированные тесты, можно отказаться от специалистов по ручному тестированию [2] [3].

Ручное тестирование проводится человеком, он играет роль конечного пользователя, на которого ориентированно разрабатываемое приложение. Рассмотрим ручное тестирование web приложений. В его рамках обычно предусматривают прямое открытие стенда с приложением и его непосредственной эксплуатации в рамках написанных сценариев. Это долгая и очень дорогая процедура, так как приложение всегда имеет достаточного широкий функционал, который необходимо проверить. И делать это нужно постоянно, так как приложение всё время находится в разработке, и любая ошибка разработчика может оказать воздействие на уже протестированный функционал.

В конечном итоге весь процесс представляет из себя множество однообразных действий, которые выполняются человеком в ручном режиме. Такие процессы в реальной жизни принято автоматизировать, тестирование не стало исключением.

Автоматизированные тесты (Автотесты) – это программа, которая способна эмулировать поведение конечного пользователя в системе, тем самым проверяя приложение, генерировать отчёты об ошибках, проделанных действиях и полученных результатах. Возможности «автотестов» очень широки и применяются повсеместно [4] [5].

Схема взаимодействия меняется как показано на рисунке 6.

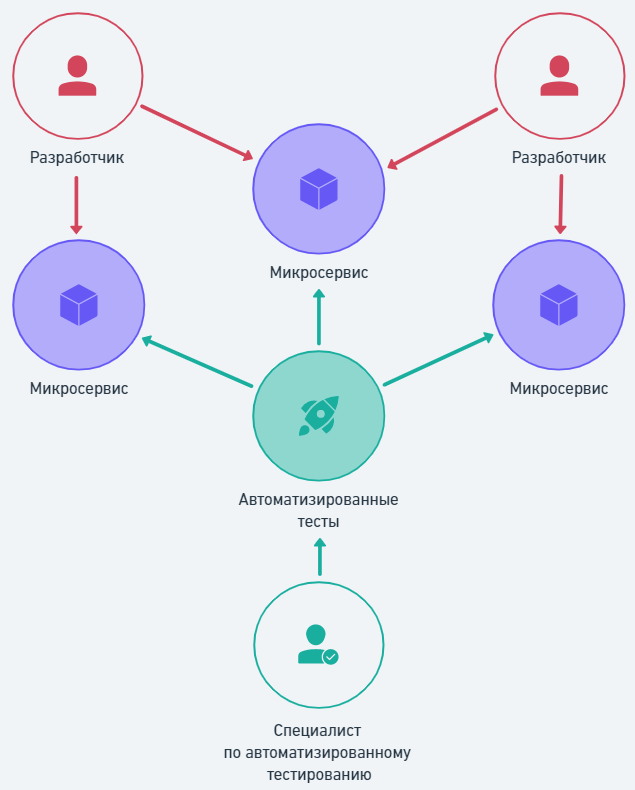


Рисунок 6 - Схема взаимодействия разработчика и автотестера №1

Достоинства:

* скорость работы намного выше, чем у человека;
* работают одновременно, в многопоточном режиме;
* отсутствие влияния человеческого фактора;
* гибкость настройки и удобство технической поддержки;
* возможность тестирования приложения целиком.

Недостатки:

* на разработку автоматизированных тестов уходит много времени;
* на больших проектах одного специалиста по автоматизированному тестированию будет недостаточно, т.к. необходимо быстро разрабатывать новые тесты.

Решение:

* нанять ещё нескольких специалистов по автоматизированному тестированию. Демонстрация решения приведена на рисунке 7.

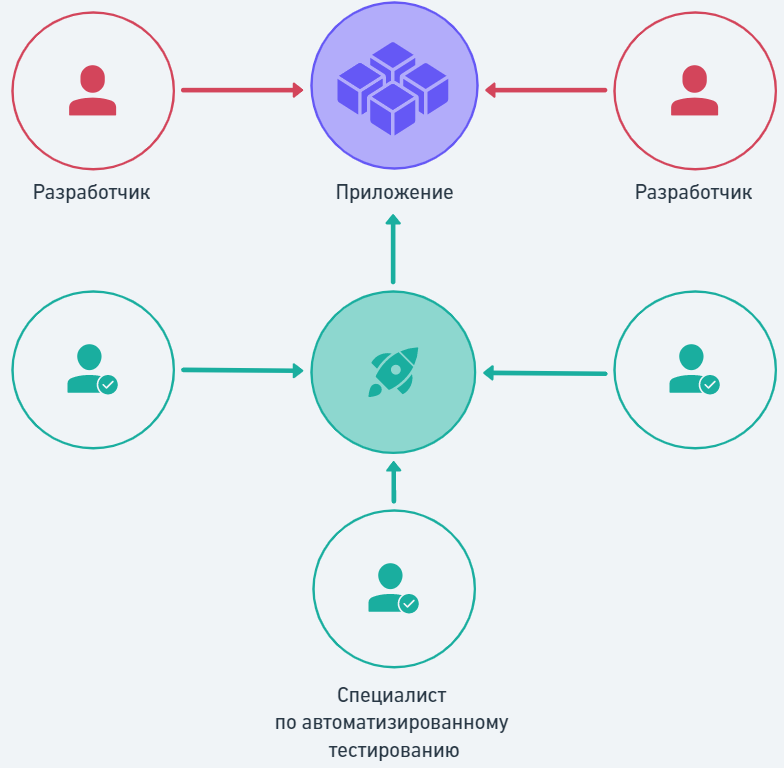


Рисунок 7 - Схема взаимодействия разработчика и авто-тестера №2

Достоинства:

* скорость работы намного выше, чем у человека;
* работают одновременно, в многопоточном режиме;
* отсутствие влияния человеческого фактора;
* гибкость настройки и удобство технической поддержки;
* возможность тестирования приложения целиком.

Недостатки:

* быстро устаревают и требуют обновления в связи с постоянными изменениями приложений;
* недоступность для специалистов по ручному тестированию;
* большие временные затраты на создание новых авто-тестов;
* каждый сценарий обрабатывается дважды.

Решение:

* комбинированный подход между ручным и автоматизированным тестированием. Внедрение новой функциональности в автоматизированные тесты, которая позволит специалистам по ручному тестированию писать автоматические тесты;
* сделать процесс тестирования программного обеспечения гибким, не ресурсоемким с точки зрения, затрачиваемого времени, вложения денежных средств и количества используемых специалистов;
* проанализировать существующие подходы к тестированию программного обеспечения выявить достоинства и недостатки;
* проанализировать пути автоматизации процесса тестирования;
* предложить подход к решению проблемы с использованием средств автоматизации.

## Разработка структуры фреймворка с гибридным подходом

Предлагается разработать приложение из трёх компонентов:

1. функциональный модуль - содержит основные функциональные возможности автоматизированного теста;
2. модуль с тестовыми сценариями - содержит тестовые сценарии;
3. связующий модуль - преобразует тестовые сценарии в полноценный автоматизированный тест.

Схематичная иллюстрация предложенных модулей представлена на рисунке 8.

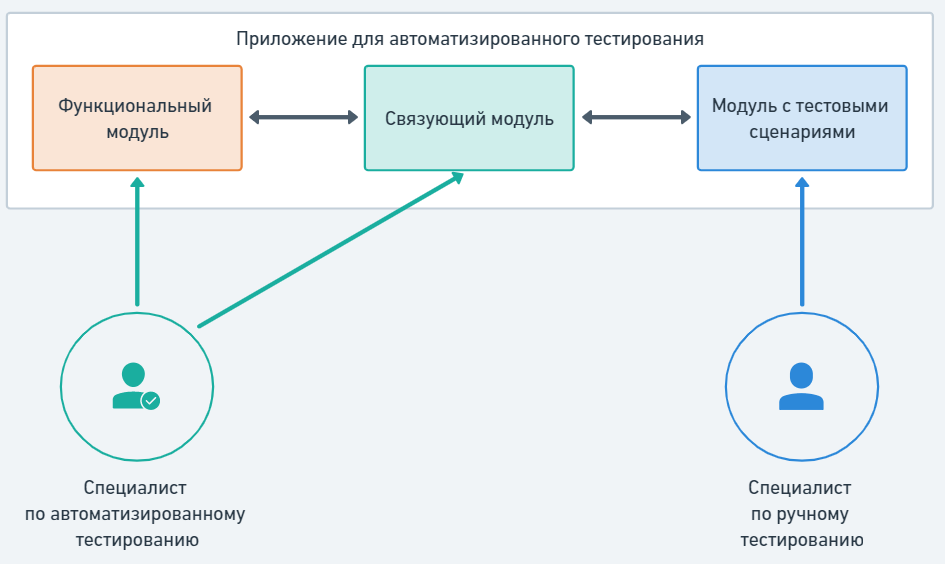


Рисунок 8 - Схема взаимодействия тестировщика и автотестера №1

Специалист по автоматизированному тестированию реализует функциональный и связующий модули, которые предоставят готовый конструктор по внедрению новых автоматизированных тестов. Обеспечит стабильную работу этих модулей и займётся их поддержкой.

Специалисты по ручному тестированию напишут тестовые сценарии для проверки работоспособности приложения. Сценарии станут основой для вызова функционального модуля, который и обеспечит работу теста.

Предлагается разработать принципиально новое решение, где специалисты по ручному тестированию смогут самостоятельно конструировать автоматизированные тесты, используя простые фразы на русском языке. Представленное решение изображено на рисунке 9.

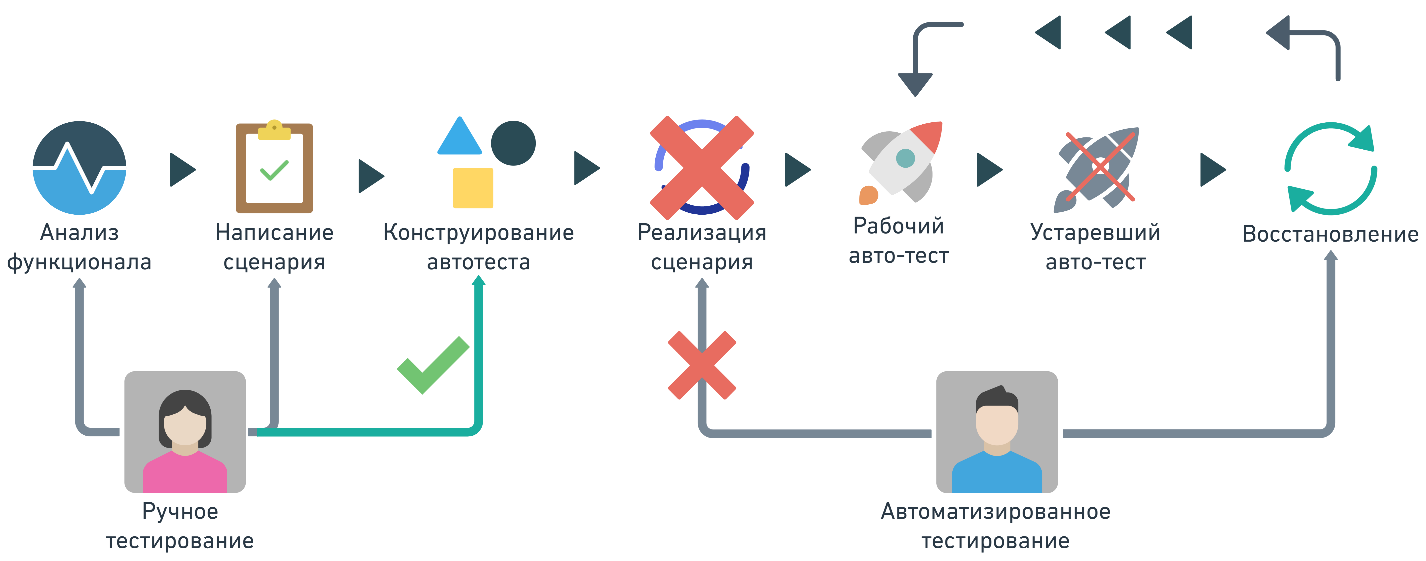


Рисунок 9 – Новое решение в создании автоматизированных тестов

Исходя из схемы, представленной выше, в дополнение к существующим решениям по проектированию автоматизированных тестов реализацией сценария теперь занят специалист по ручному тестированию. Это возможно, благодаря конструктору, который был внедрён в новом решении.

Такой подход позволяет сразу из сценария получать рабочие авто-тесты и избавится от большей части недостатков, сделав процесс тестирования максимально комфортным, гибким и эффективным.

Текущий подход к тестированию ПО позволит специалистам по ручному тестированию сразу конструировать автоматизированные тесты, а специалистам по автоматизированному тестированию заниматься расширением функционала.

### Архитектура модулей гибридного фреймворка

Каждый модуль является самостоятельной единицей и способен встраиваться в любые приложения как в комбинации с другими, так и индивидуально.

Модуль с тестовыми сценариями подразумевает под собой набор тестов, написанных для всех предыдущих релизов, образующих тесты типа регрессионные и набор для нового релиза. Архитектура модуля представлена на рисунке 10.

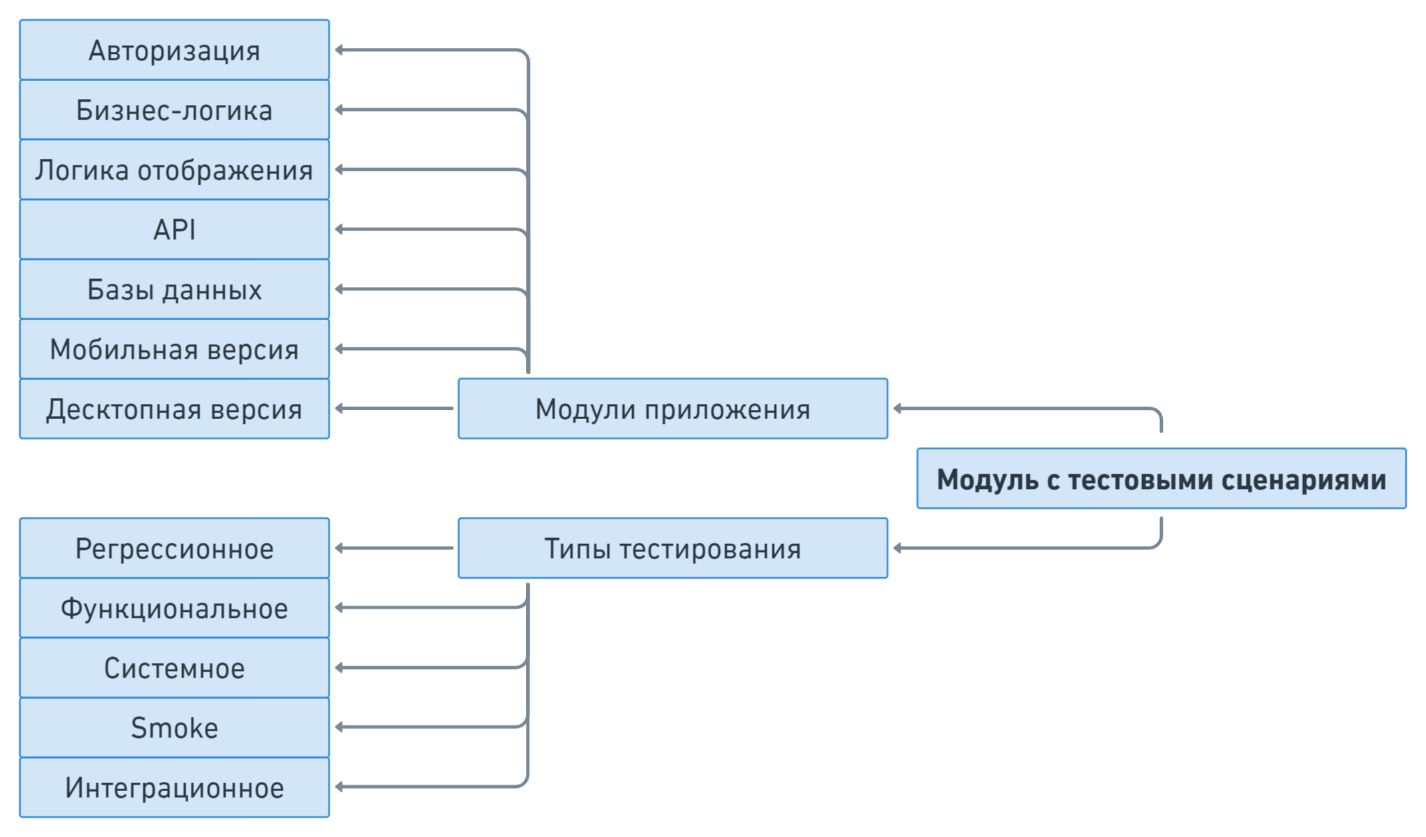


Рисунок 10 - Структурная схема модуля с тестовыми сценариями

Модуль тестовых сценариев устроен так, чтобы в любой момент времени можно было легко добавить новые тесты, а также изменить уже написанные, если нужно. Все тесты старого релиза объединяются в общие подтип регрессионные и запускаются на ежедневной основе, обычно в ночное время.

Все тесты подразумевают под собой обычный текстовый файл со специальным расширением, написанный на русском языке. Там содержатся основные шаги сценария, включая все подготовительные и завершающие действия.

Для того, чтобы добавить новые тесты необходимо просто добавить нужный сценарий к уже написанным тестам, либо создать новый тест, в котором описать нужный функционал. Далее связующий модуль преобразует текст в файле в программный вызов метода.

Связующий модуль представляет собой жесткую связь слово – метод, в которой словом является шаг в тестовом сценарии, а методом фрагмент реального программного кода.

Каждый файл с тестовым сценарием необходимо описать специальным объектом с помощью кода для корректной обработки. Связующий модуль в данной реализации выполнен на основе построчного считывания файла сценария и дальнейшего вызова программного кода. Архитектура модуля представлена на рисунке 11.

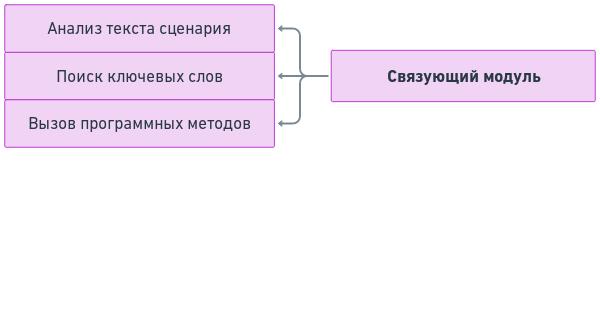


Рисунок 11 - Структурная схема связующего модуля

Функциональный модуль представляет из себя полноценное приложение для создания автоматизированных тестов, со своей архитектурой, паттернами и программной реализацией.

Данный модуль реализуется силами специалистов по автоматизированному тестированию, они несут за него ответственность и заняты полным сопровождением.

В общем виде структура функционального модуля изображена на рисунке 12.

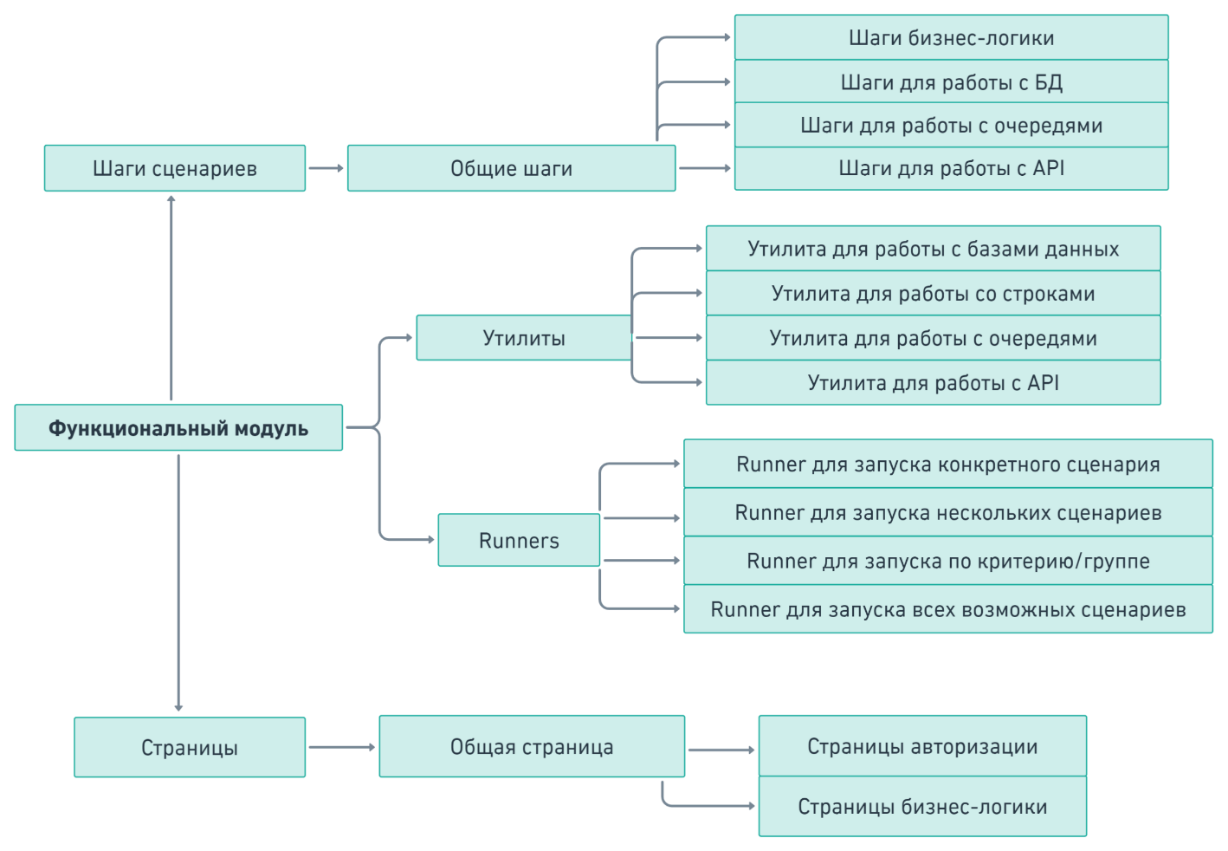


Рисунок 12 - Структурная схема функционального модуля

Ключевым отличием приложения для авто-тестов является его универсальность с точки зрения решения задачи тестирования программного обеспечения, дело в том, что тестировать получиться не только web приложения, но и их API, базы данных, очереди и др. Поэтому подходов к реализации огромное количество, но выделить некоторые паттерны можно [5][6].

Функциональный модуль делится на четыре больших раздела, каждый из которых отвечает за определённый функционал. Страницы отвечают за элементы, которые на них отражаются, утилиты – это класс-помощники которые упростят работу с базовым функционалом в определённых областях тестирования. Шаги сценариев – это действия, которые можно выполнять специалистам по ручному тестированию в рамках самих сценариев. Runners – это классы, которые позволяют запускать только некоторый обособленный по признакам набор тестов.

Общие шаги представляют собой реализацию шаблонов, действий, доступных специалистам по ручному тестированию. Другими словами, это полноценный интерфейс для управления браузером, который способен эмулировать поведение пользователя в любых комбинациях. Есть возможность отрывать браузер, нажимать на кнопки, искать элементы на странице, перетаскивать их, скачивать и обрабатывать файлы.

Утилиты позволят работать с внешними источниками, например, с базами данных, API, очередями или хранилищами используя функциональный подход. Другими словами, утилиты призваны упростить рутинные операции специалиста по автоматизированному тестированию.

Страницы содержат классы с ключевыми элементами, которые отображаются на HTML странице тестируемого web приложения. Они служат для того, чтобы обернуть сложные XPATH выражения в логические названия элементов для упрощения доступа к ним и множественного использования в прочих классах [5][6].

В общем виде целевая архитектура фреймворка представлена на рисунке 13.

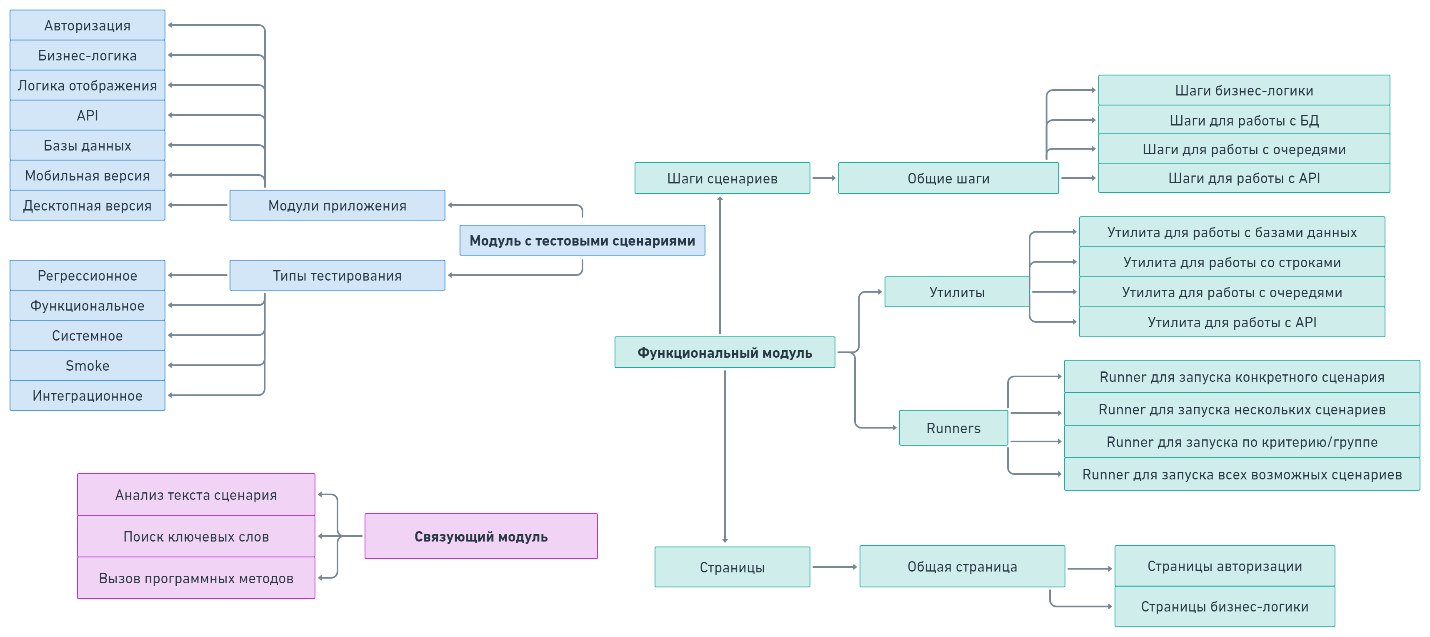


Рисунок 13 – Целевая архитектура гибридного фреймворка.

Изначально сценарии загружаются в модуль тестовых сценариев, каждый сценарий распределяется по типу проводимого тестирования или по функциям тестируемого приложения.

Далее связующий модуль проводит анализ, делает поиск ключевых слов и соединение шаблонов со сценариями построчно. И в конечном итоге совершает программный вызов функционального модуля.

Функциональный модуль разбит на множество подфункций для упрощения работы с большим объёмом шаблонов и возможных шагов внутри сценариев.

Таким образом, новое решение содержит в себе не только все названные преимущества автоматизированных тестов, но и решение множества проблем в существующих фреймворках, таких как:

* наличие программного конструктора для создания авто-тестов;
* возможность предоставления сценария на русском языке;
* исходный сценарий пишется только один раз;
* освобождение человеческих ресурсов для написания авто-тестов;
* предоставление возможности конструирования авто-теста специалистам без знания языков программирования.

### Шаблоны для унификации формирования тестовых сценариев

Вводятся следующие термины.

Тестовый сценарий – это последовательность действий, выполняемая над проверяемым программным обеспечением с целью выявить недостатки.

Шаблон (метод) – обобщённое действие, которое можно произвести в рамках тестового сценария.

Универсальность гибридного фреймворка заключается в формировании большого количества шаблонов (методов), которые позволят осуществить множественное использование одного и того же функционала в нескольких сценариях, тем самым обеспечивая гибридность и функциональный (нативный) подход в реализации сценариев и авто-тестов одновременно.

Для написания шаблонов, прежде всего, потребуется изучение существующего функционала тестируемого приложения. Полное покрытие шаблонами тестируемого web приложения займёт гораздо меньше времени, чем описание каждого сценария вне конструктора, потому данный подход более рентабельный в промышленных масштабах.

Специалист по автоматизированному тестированию анализирует функциональность приложения и опираясь на заявленный функционал начинает описывать шаблоны программными методами.

В конечном итоге специалисты по ручному тестированию получат полноценный функционал для работы с ПО.

### Алгоритмы работы в модулях

Все основные алгоритмы программы заключены в функциональном модуле. Для работы со страницами браузера необходимо реализовать алгоритм поиска элементов.

Страница HTML представляет собой дерево объектов, у каждого из которых могут быть свои дочерние объекты. Иначе это называется «Document Object Model» (DOM).

XPath селекторы используются для быстрой навигации по элементам страницы, они представляют из себя полный путь или относительный путь до необходимого объекта, также у них есть множество специальных функций, например, функция «text» возвращает текст элемента.

Квадратные скобки означают порядковый номер вложенного элемента.

По мимо XPath существуют CSS селекторы, они считаются более надёжными из-за редко меняющихся стилей на страницах приложений. Оба алгоритма поиска реализованы через обход дерева и считывания каждого элемента на пути.

Алгоритм продемонстрирован на рисунке 14.



Рисунок 14 – Алгоритм поиска элементов DOM

Исходя из блок-схемы можно сделать вывод, что XPath/CSS селекторы работают по одному и тому же сценарию. Главное и весомое отличие CSS от XPath в том, что CSS селекторы не работают с текстом элементов, что делает их менее универсальными.

Все алгоритмы поиска элементов базируются на обходе дерева DOM, однако существуют некоторые функции селекторов, которые позволяют существенно сократить время поиска. Например, мы можем указать конкретный атрибут элемента, чтобы механизмы языка загружали только ту часть страницы, в которой они существуют. Этот механизм сократит время на ожидании загрузки страницы в буфер [5][6].

Остальные алгоритмы работы уникальны для каждого отдельно случая, однако в большинстве своём базируются на поиске элементов и дальнейшей их обработке.

Алгоритм составления тестовых шагов относится к модулю тестовых сценариев и является уникальным в каждом отдельном случае. Однако можно составить некоторую обобщённую схему для составления любого сценария, представленную на рисунке 15.



Рисунок 15 – Алгоритм написания тестового сценария

Любой тестовый сценарий инициирован введением нового функционала в приложение. Далее написание сценария зависит от выбранного класса [5][6].

Сценарий может состоять из следующих блоков.

Шаги приготовления. Используется, если в сценарии необходимы какие-либо приготовления, например – установить соединение с базой данных, открыть браузер и подключиться к Apache Kafka. Необходимы, если множество тестов требуют одних и тех же исходных условий

Исходные данные. Используется, если в сценарии задействованы исходные данные, такие как список пользователей, платежей, покупок и т.д. Может быть один тестовый набор исходных данных для множества сценариев.

Шаги теста. Используются для формирования набора действий конкретного сценария. Обычно уникальны в рамках одного теста, однако могут быть частично переиспользованы в других сценариях.

Ожидаемый результат. Результат, который ожидается согласно техническому заданию. Может быть частью блока исходных данных.

Проверка. Получение результата непосредственно из приложения и сравнение его с ожидаемым результатом. Либо иная проверка работы приложения.

Шаги завершения сценария. Используется, если сценарий не сохраняет консистентность приложения.

Связующий модуль не имеет алгоритмов, он лишь позволяет связать файлы с методами программного кода построчным считыванием.

В качестве примера тестируемого приложения для реализации предложенных модулей используется простой интернет магазин, в котором есть всего несколько основных возможностей – это авторизация, выбор товара, оформление заказа и покупка.

### Реализация модулей

#### Реализация модуля тестовых сценариев

Модуль тестовых сценариев предлагается реализовать через структуру папок и файлов внутри проекта с исходным кодом. Архитектура построения папок будет подразумевать разбиение сценариев по логическим частям тестируемого приложения. Каждая папка будет содержать в себе сценарии исключительно определённой части функционала приложения.

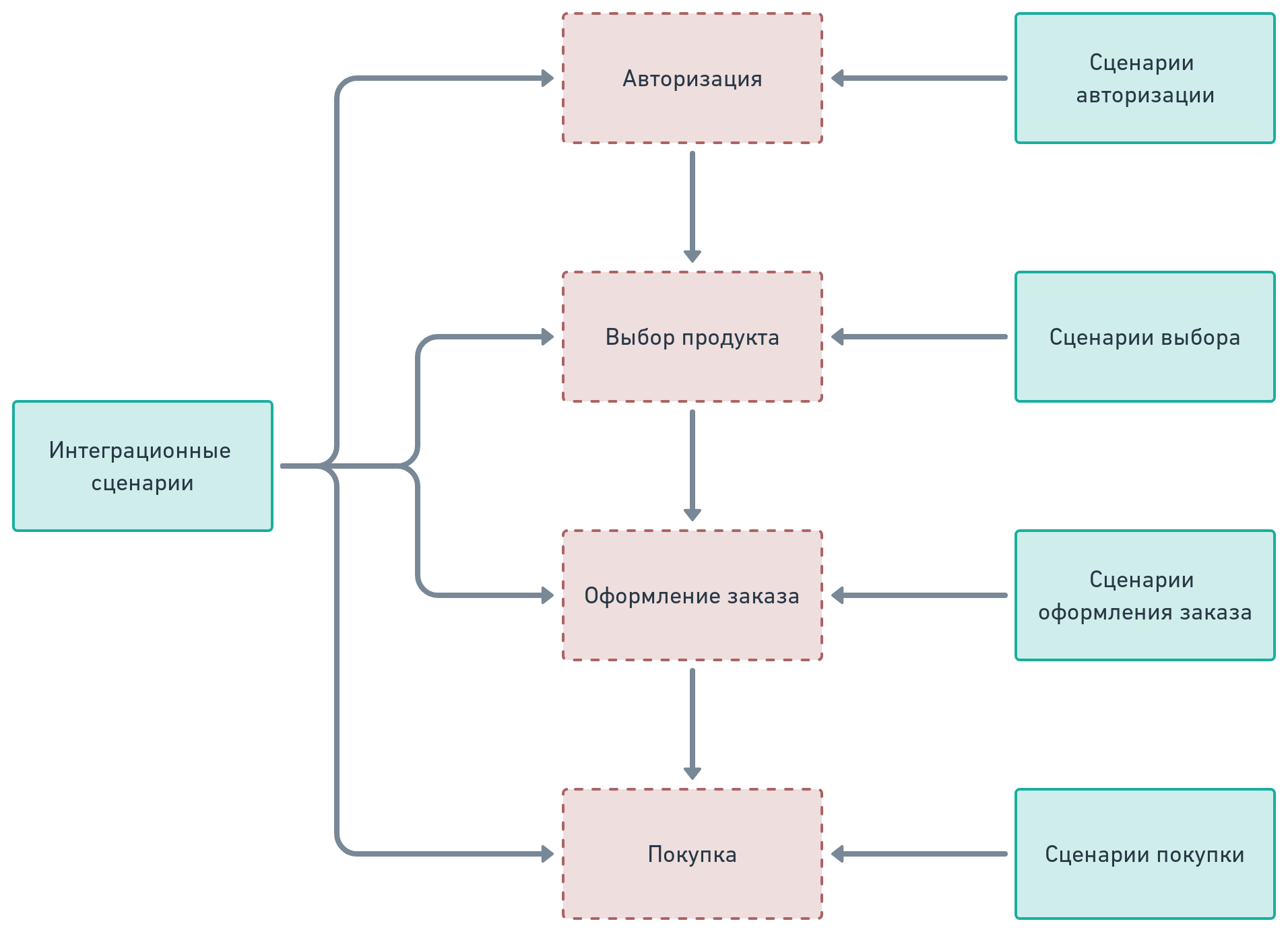


Рисунок 16 – Структура сценариев для интернет-магазина

Из рисунка 16 видно, что на каждое возможное действие пользователя у нас имеются, как и отдельные сценарии, так и полные маршруты по всему функционалу.

Для этого создадим специальные директории, в которые разместим файлы с тестовыми сценариями. Схема изображена на рисунке 17.

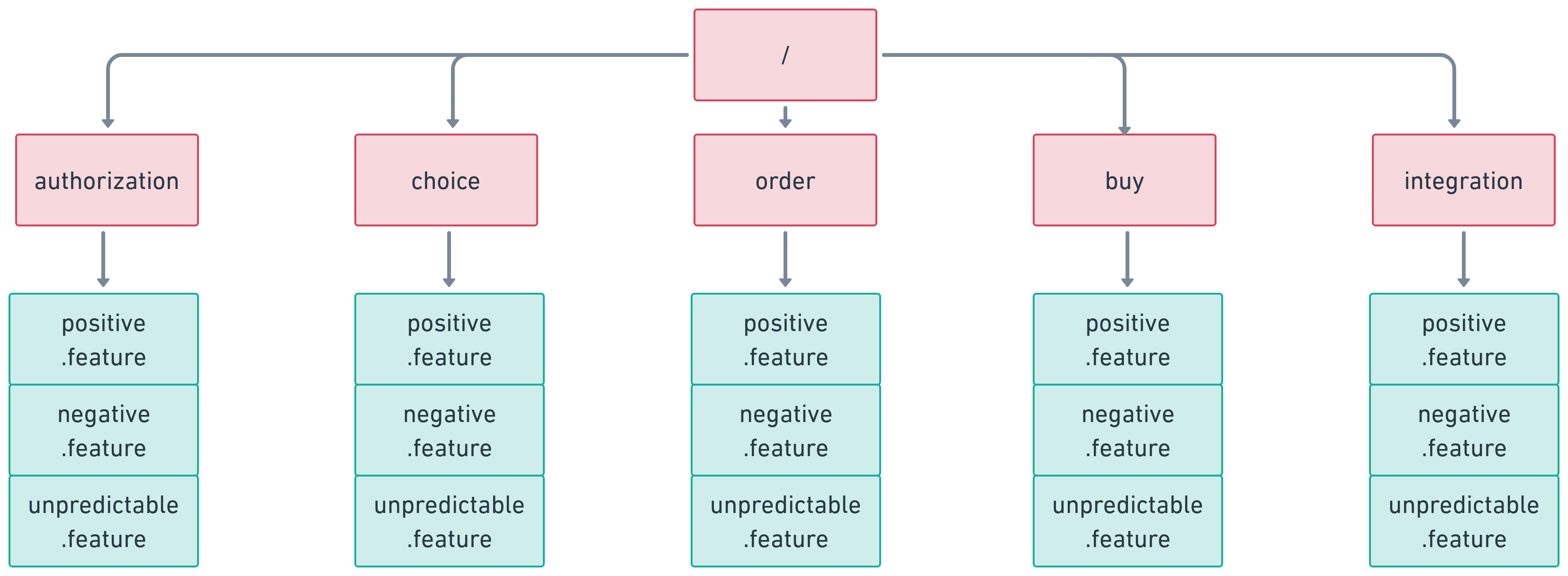


Рисунок 17 – Структура папок и файлов

В корне схемы находится исходный каталог, может быть любым. Красным цветом обозначены директории, зеленым цветом файлы в формате feature.

Такое разбиение позволит быстро масштабировать сценарии при их огромном количестве и легко ориентироваться в нужных элементах тестируемого приложения. Наглядно можно увидеть, что уже покрыто тестами, а что находится в разработке. Разделение тестов на папки является хорошей практикой в проектировании архитектуры тестовых сценариев.

Приведем пример тестового сценария на рисунке 18.



Рисунок 18 – Файл с примером тестового сценария

Специальное слово «Feature» служит для объявления набора тестов, у него есть описание и название.

Слово «Scenario» означает объявление самого теста, имеет имя.

Слово «Then» означает шаг внутри теста, например, «Нажать на кнопку подтверждения» значит найти на странице кнопку типа «submit» и кликнуть на неё.

Ключевые слова позволяют однозначно определить, что является шагом теста, а что комментарием или описанием. Они служат для связи сценариев с функциональным модулем.

#### Реализация связующего модуля

Связующий модуль реализован при помощи библиотек компании «Cucumber», в нём задействован функционал связи ключевых слов с кодом проекта. Реализация основана на имплементации ключевых слов в файлах feature и их связи с реальными тестовыми методами через аннотации @Then.

Cucumber ищет среди классов проекта методы помеченные аннотацией @Then и исполняет их, передавая необходимые аргументы или без таковых.

Стоит дополнить, что связующий модуль является лишь удобным решением для связи текста с программным кодом, никакой другой логики он не подразумевает, потому реализация этого модуля не было изменена в рамка выпускной квалификационной работы.

Таким образом, связующий модуль используется исключительно как сторонняя библиотека для реализации связи между описанием тестового сценария и исполнением реального кода проекта.

#### Реализация функционального модуля

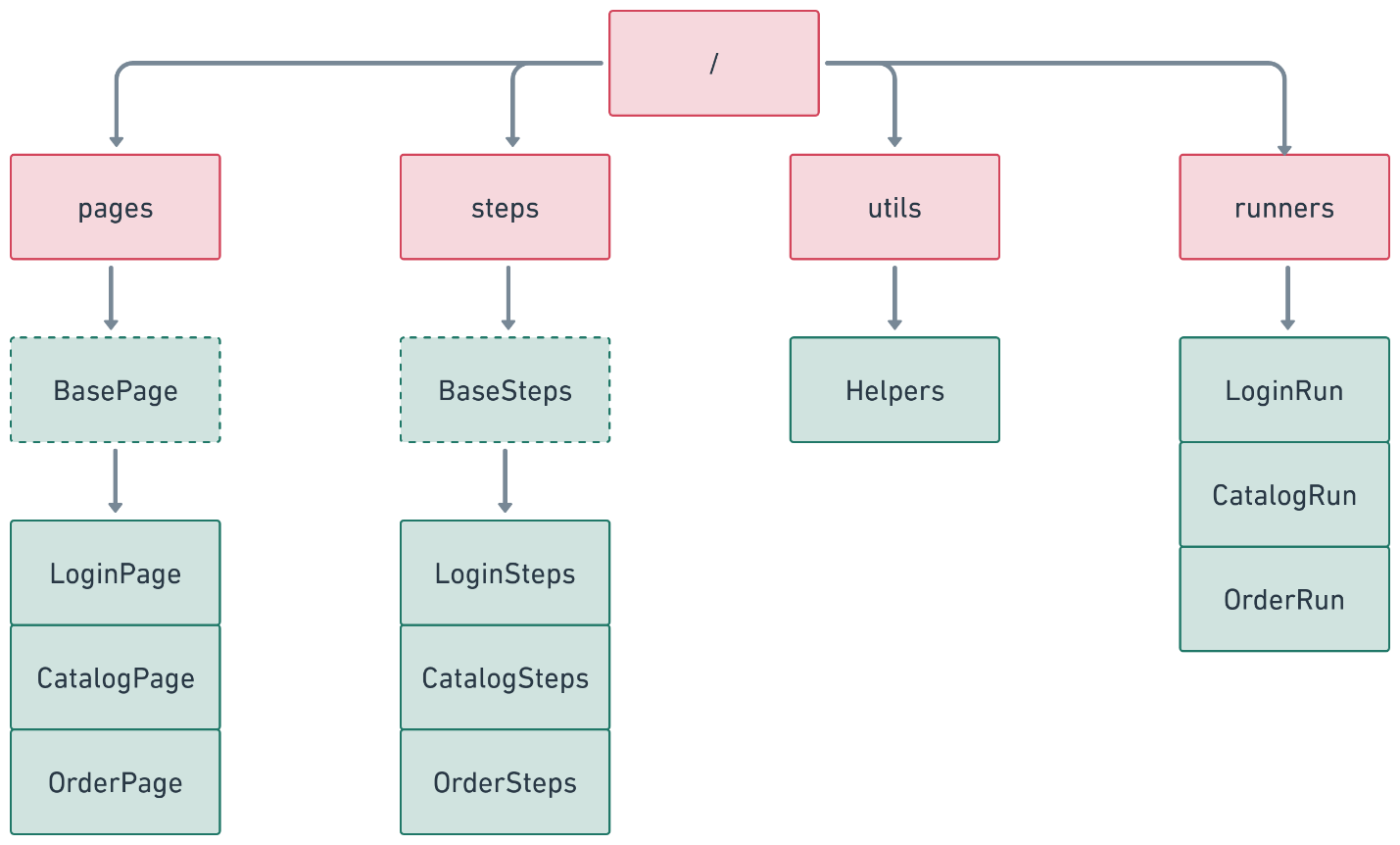


Рисунок 19 – Структура классов и директорий

В корне схемы на рисунке 19 находится исходный каталог, может быть любым. Красным цветом обозначены директории, зеленым цветом программные классы. Абстракции изображены пунктиром.

Директория pages содержит в себе классы, описывающие элементы web страниц и действия, которые можно производить над ними. Она служит для предоставления программисту набора методов и функций для работы с конкретной страницей тестируемого приложения

Директория steps содержит классы реализующие тестовые шаги, описанные в конкретных сценариях. Шаги являются конечной абстракцией в тестах, именно в них находится самая низкоуровневая логика теста.

Директория runners содержит классы, отвечающие за запуск тестов и сканирование определённого сценария из выбранного файла feature.

Директория utils содержит вспомогательные классы - утилиты, которые упрощают работу со специфичными инструментами.

### Пример применения типовой модели автоматизированных тестов

В качестве контрольного примера предлагается рассмотреть простое web приложение (интернет-магазин), на которое будут написаны тесты с использованием архитектуры описанной в предыдущих главах.

Тестируемое приложение имеет следующий функционал:

* авторизация пользователя;
* просмотр продукта;
* выбор товара из каталога;
* наполнение корзины с продуктами;
* оформление заказа.

Для начала необходимо написать тестовый сценарий, который далее будет реализован специалистами по автоматизированным тестам.

Составим простой сценарий для проверки возможности оформления заказа с одним товаром, изображен на рисунке 20.

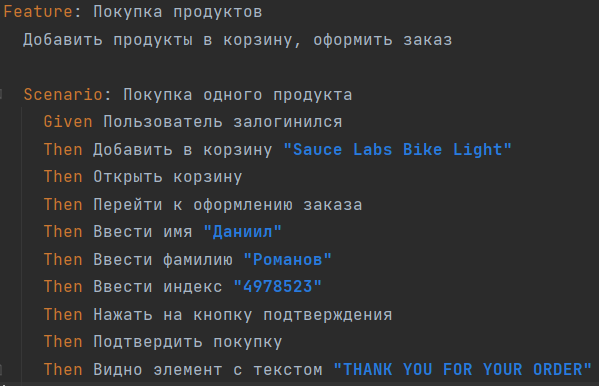


Рисунок 20 – Сценарий оформления заказа с одним товаром

Данный сценарий содержит предусловие к выполнению теста, пользователь должен пройти авторизацию перед совершением покупки.

Далее происходит добавление товара в корзину, заполнение данных по доставке, оплата и оформление.

Реализация сценария подразумевает под собой реализацию каждого шага, описанного в сценарии. Например, реализуем авторизацию на главной странице, изображённой на рисунке 21.



Рисунок 21 – Страница авторизации

На странице с авторизацией используется всего три элемента, это поле ввода имя пользователя, поле ввода пароля и кнопка логина. Соответственно нам необходимо реализовать Java класс, который опишет все необходимые методы для работы с этой страницей, а именно два поля для ввода логина и пароля.

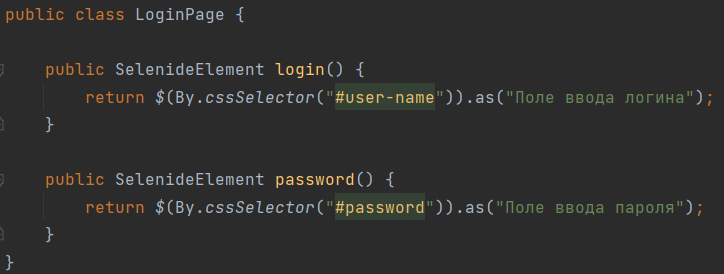


Рисунок 22 – Класс LoginPage

На рисунке 22 в классе LoginPage не реализован метод для поиска кнопки подтверждения из-за того, что она не является уникальной и присутствует на множестве других страниц. Поэтому она реализована в другом классе.

В этом классе содержатся методы, которые возвращают элементы DOM в виде объектов класса SelenideElement, у которых есть множество методов по управлению, таких как click (нажать на элемент), value (ввести значение) и т.д.

Поиск элементов в данном случае завязан на CSS селекторах и реализован через поиск элемента по уникальному идентификатору, что является самым надёжным способом найти элемент на странице.

Реализуем методы для ввода логина и пароля по отдельности, так как скорее всего они нам понадобятся в дальнейших тестах, например, при тестировании страницы с авторизацией. Создадим класс для реализации шагов, внутри которого создадим объект класса LoginPage и с его помощью опишем шаги по вводу логина и пароля, представленные на рисунке 23.

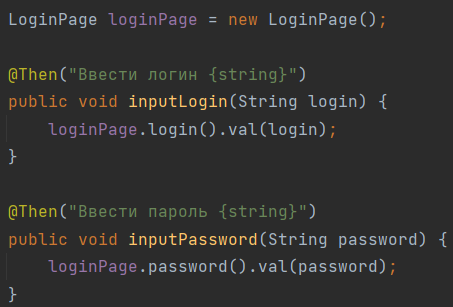


Рисунок 23 - Класс LoginSteps

Реализация шагов по вводу логина и пароля осуществляется с помощью метода val, который позволяет вводить текст в элементы DOM типа input. Таким образом, после реализации этих шагов мы можем приступить к реализации предусловия данного сценария.

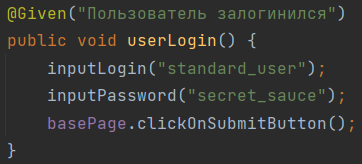


Рисунок 24 – Метод userLogin

На рисунке 24 добавляется общий метод по нажатию кнопки подтверждения, его реализация содержится в классе BaseSteps.

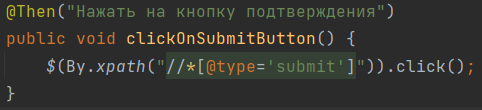


Рисунок 25 – Метод clickOnSubmitButton

Метод на рисунке 25 ищет все элементы с типом Submit, реализация происходит средствами XPath, и сразу же вызывается метод click, т.к. никаких других действий с кнопкой подтверждения сделать невозможно.

Таким образом реализуются все оставшиеся шаги сценария. Главное, что необходимо учитывать при реализации – это возможность использования написанного кода в будущем, следуя этим принципам получится покрыть потребности любого возможного сценария.

Время выполнения данного сценария составило 5.744 секунды, по окончании сценария генерируется отчёт о проделанной работе, представленный на рисунке 26, в котором представлены шаги, время их выполнения, описание каждого шага и т.д.

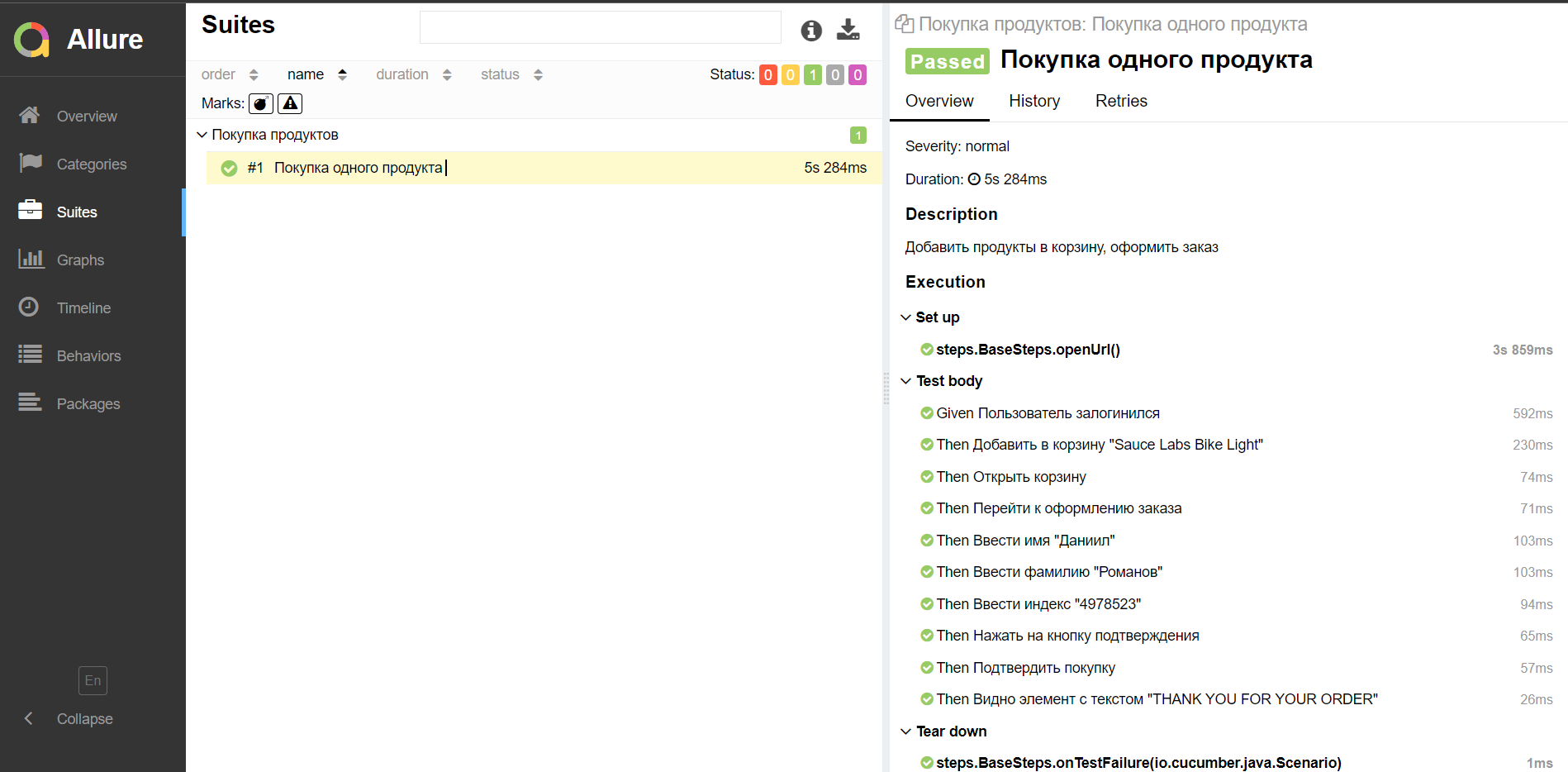


Рисунок 26 – Сгенерированный отчёт Allure2

В отчёте представлен каждый шаг, время его выполнения, предусловия и постусловия выполнения теста. Соответственно при запуске множества тестов мы получаем отчёт о выполнении каждого теста. Если происходит ошибка, то она выводится в отчёт, представленный на рисунке 27, к нему прикрепляется снимок экрана, чтобы специалист по ручному тестированию мог легко разобраться в причинах ошибочного поведения. Для демонстрации изменим последний шаг и будем ожидать элемент, которого нет на экране.

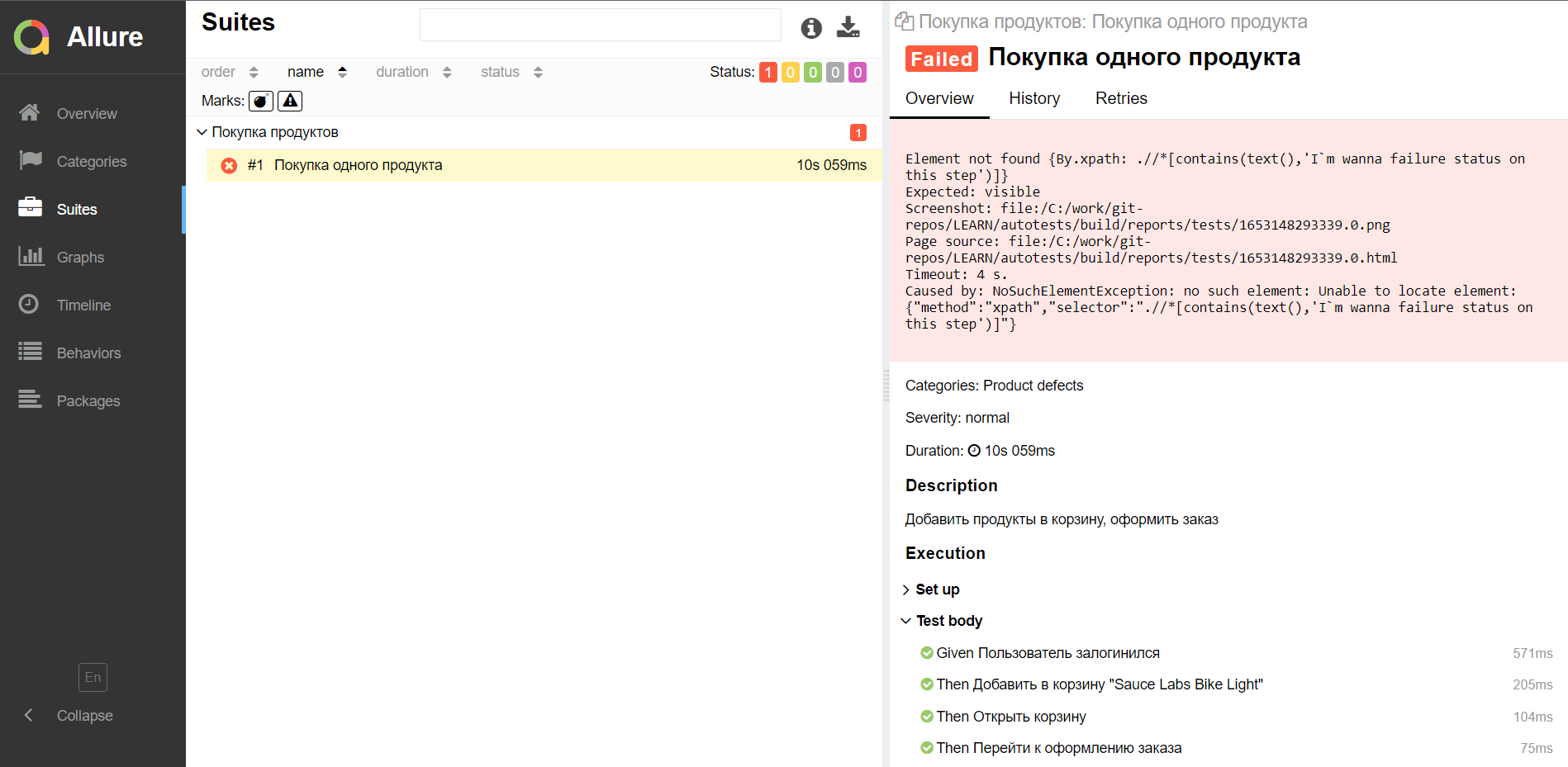


Рисунок 27 – Отчёт об ошибках

Сверху отчёта представлены технические ошибки приложения, сам тест перешёл в статус «Failed», и перекрасился в красный цвет для наглядности.

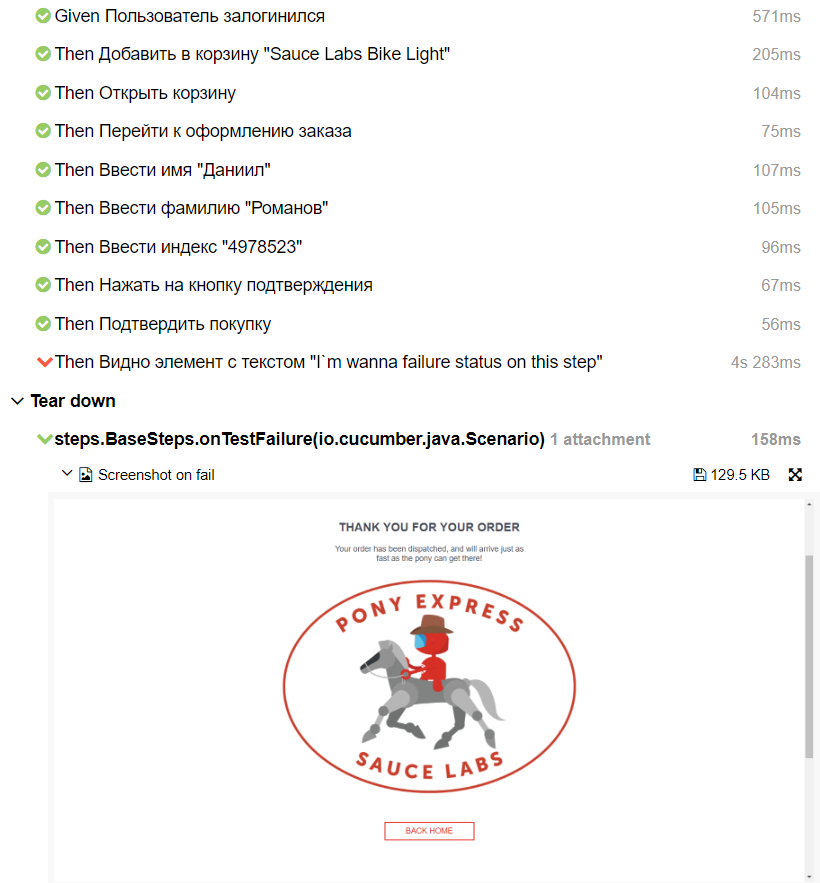


Рисунок 28 – Отчёт об ошибках

Внизу отчёта на рисунке 28 можно увидеть, какой конкретно шаг был провален и прикреплённый к нему снимок экрана.

По отчёту ясно, что ожидался текст «I`m wanna failure status on this step», а на самом деле был изображен текст «THANK YOU FOR YOUR ORDER».

Таким образом, целевое решение для автоматизации заключается в том, что теперь писать новые автоматизированные тесты становится намного проще и быстрее, что существенно ускоряет внедрение.

## Оценка юзабилити разработанного фреймворка

### Метод экспертных оценок

Сущность метода экспертных оценок заключается в проведении экспертами интуитивно-логического анализа проблемы с количественной оценкой суждений и формальной обработкой результатов. Получаемое в результате обработки обобщенное мнение экспертов принимается как решение проблемы.

Характерными особенностями метода экспертных оценок как научного инструмента решения сложных неформализуемых проблем являются:

* научно обоснованная организация проведения всех этапов экспертизы, обеспечивающая наибольшую эффективность работы на каждом из этапов;
* применение количественных методов как при организации экспертизы, так и при оценке суждений экспертов и формальной групповой обработке результатов.

Особой разновидностью метода экспертной оценки является экспертный опрос — разновидность опроса, в ходе которого респондентами являются эксперты — высококвалифицированные специалисты в определенной области деятельности.

Эксперт – это компетентное лицо, имеющее глубокое знания о предмете или объекте исследования.

Метод подразумевает компетентное участие специалистов в анализе и решении рассматриваемой проблемы.

Для оценки приложения была отобрана группа сотрудников, которые являлись пользователями этого приложения. Ниже сформирован список критериев, подлежащих оценке:

### Оценка юзабилити разработанного фреймворка

Для оценки юзабилити нового фреймворка воспользуемся критериями, представленными в таблице 1.

Таблица 1 – Список критериев

|  |  |  |
| --- | --- | --- |
| № критерия | Показатель | Описание |
| 1 | Доступность | Простота написания нового сценария |
| 2 | Информативность отчётов | Информативность отчётов об ошибках |
| 3 | Поддержка | Простота поддержки фреймворка |

В качестве экспертов выступили:

Эксперт 1 – Жуков Андрей Сергеевич, специалист по автоматизированному тестированию, ГК «Иннотех»

Эксперт 2 – Протопопов Артём Борисович, ведущий инженер программист, ГК «Иннотех»

Эксперт 3 – Голубович Михаил Валерьевич, ведущий специалист по автоматизированному тестированию, «Edgify Ltd.»

Эксперт 4 – Фирсов Евгений Олегович, ведущий специалист по автоматизированному тестированию, ПАО «Сбербанк»

Эксперт 5 – Соловьев Андрей Александрович, ведущий специалист по ручному тестированию, ГК «Иннотех»

Оценивать важность каждого критерия необходимо было группе экспертов – пользователей данного приложения. Каждый член группы должен был проранжировать список ранее указанных критериев, с соблюдением следующих условий.

Для оценки можно использовать три ранга:

* ранг «1» присуждается самому значимому критерию среди других
* ранг «3» присуждается критерию, имеющему наименьшее значение среди других
* эксперт может присуждать нескольким критериям одинаковые ранги
* каждый эксперт обособлен от других экспертов

Таблица 2 – Ранжирование критериев

|  |  |  |  |
| --- | --- | --- | --- |
| Ранжирование | № критерия | | |
| 1 | 2 | 3 |
| Эксперт 1 | 3 | 2 | 1 |
| Эксперт 2 | 1 | 2 | 1 |
| Эксперт 3 | 1 | 1 | 2 |
| Эксперт 4 | 2 | 1 | 2 |
| Эксперт 5 | 1 | 2 | 1 |

Проведем обработку результатов ранжирования критериев с целью поиска коэффициентов важности. Если ранги у какого-либо критерия совпадают, необходимо привести ранжировки к нормализованному виду (в соответствие ставится ранг, вычисляемый путем поиска среднего значения номеров мест критериев в ранжировке). Так как в строках Таблицы 2 имеются совпадающие значения рангов (например, строки 2, 3, 4, 5), то приводим оценки экспертов к нормальному виду в Таблице 3.

Таблица 3 – Нормализованная таблица ранжирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Норм. ранжирование | № критерия | | | Сумма |
| 1 | 2 | 3 |
| Эксперт 1 | 3 | 2 | 1 | 6 |
| Эксперт 2 | 1,5 | 3 | 1,5 | 6 |
| Эксперт 3 | 1,5 | 1,5 | 3 | 6 |
| Эксперт 4 | 2,5 | 1 | 2,5 | 6 |
| Эксперт 5 | 1,5 | 3 | 1,5 | 6 |
|  | 10 | 10,5 | 9,5 | 30 |

### Проверка правильности нормализации ранжировок

Для проверки правильности нормализации ранжировок вычислим сумму рангов для i-го эксперта по формуле 1. Результаты вычислений представлены в формуле 2.

где n – количество экспертов.

В таблице 2 сумма ранжировок в каждой строке равна 6 и сходится с полученной в результате расчетов, следовательно, можно сделать вывод о том, что нормализация ранжировок выполнена верно.

### Вычисление коэффициентов важности критериев

Оценим важность каждого критерия для экспертов по формуле 3.

где m – количество критериев;

n – количество экспертов;

– сумма ранжирования по каждому критерию.

Из расчетов, приведенных в формулах 4, 5, 6, следует, что для данной группы экспертов порядок важности критериев следующий:

1. поддержка (оценка того, насколько просто будет поддерживать фреймворк);
2. доступность (оценка простоты использования фреймворка);
3. информативность отчётов (оценка того, насколько информативными получаются отчёты о тестировании).

### Балльная оценка качества разработанного фреймворка

Далее экспертам было предложено оценить функциональность приложения с использованием бальной системы. Оценки от пяти экспертов приведены в Таблице 4.

Обозначения оценок:

5 – «Отлично»

4 – «Хорошо»

3 – «Допустимо»

2 – «Плохо»

Таблица 4 – Оценки экспертов по каждому критерию

|  |  |  |  |
| --- | --- | --- | --- |
| Оценки | № критерия | | |
| 1 | 2 | 3 |
| Эксперт 1 | 5 | 5 | 5 |
| Эксперт 2 | 5 | 5 | 4 |
| Эксперт 3 | 5 | 5 | 4 |
| Эксперт 4 | 5 | 4 | 5 |
| Эксперт 5 | 4 | 4 | 5 |
| Ср. оценка | 4,8 | 4,6 | 4,6 |

### Расчёт оценки качества разработанной системы

где n – количество экспертов;

– средняя оценка эксперта;

– важность (вес) критерия.

В результате оценки качества по формуле 7, была получена следующая оценка: 4,667. Вопрос о внедрении фреймворка в промышленную эксплуатацию остаётся за специалистами по ручному и автоматизированному тестированию.

# ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы был произведен анализ существующего процесса проведения автоматизированного тестирования в крупных компаниях, были выявлены узкие места процесса и предложен вариант их устранения, обозначены задачи, подлежащие исполнению, разработан гибридный фреймворк для проведения автоматизированного тестирования.

Разработанный фреймворк имеет широкий функционал, позволяет проводить автоматизированное тестирование.

Были решены следующие задачи:

1. предложен подход для решения проблем автоматизации тестирования;
2. разработана архитектура модулей гибридного фреймворка;
3. разработаны шаблоны для унификации формирования тестовых сценариев;
4. разработан удобный и доступный для специалистов по ручному тестированию конструктор автоматизированных тестов;
5. представлена типовая модель автоматизации тестирования на контрольном примере.

Проведён анализ разработанного программного обеспечения с целью установить рентабельность его применения в промышленной эксплуатации, получены высокие оценки от разных экспертов в области автоматизированного и ручного тестирования.

По итогам выполнения выпускной квалификационной работы гибридный фреймворк было решено внедрить в промышленную эксплуатацию и использовать сначала в качестве пилотного решения, далее в качестве целевого по всей компании.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Овчинникова М. А., Овчинников М. А. ПРИМЕНЕНИЕ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ В ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЯХ //Ответственный редактор. – 2022. – С. 128.
2. Бойко В. А., Легалов А. И., Зыков С. В. Архитектура интеллектуальной системы тестирования //Журнал Сибирского федерального университета. Техника и технологии. – 2022. – Т. 15. – №. 2. – С. 274-282.
3. Милов В. Р. и др. ПОДХОД К РАЗРАБОТКЕ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ТЕСТИРОВАНИЯ ЗНАНИЙ НА ОСНОВЕ СЕМАНТИЧЕСКОГО АНАЛИЗА СВОБОДНЫХ ОТВЕТОВ //Инновационные технологии в образовательной деятельности. – 2022. – С. 109-112.
4. Силаков Д. В. Автоматизация тестирования web-приложений, основанных на скриптовых языках //Труды Института системного программирования РАН. – 2008. – Т. 14. – №. 2. – С. 159-178.
5. Медведев Ю. С., Терехов В. В. Автоматизация функционального тестирования web-приложений //Современные наукоемкие технологии. – 2016. – №. 5-1. – С. 58-61.
6. Рахманов А. С., Тутубалин П. И. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ БЕЗОПАСНОСТИ ВЕБ-ПРИЛОЖЕНИЙ //Молодежь и наука: актуальные проблемы фундаментальных и прикладных исследований. – 2021. – С. 298-301.

# ПРИЛОЖЕНИЕ А

Feature: Покупка продуктов

Добавить продукты в корзину, оформить заказ

Scenario: Покупка одного продукта

Given Пользователь залогинился

Then Добавить в корзину "Sauce Labs Bike Light"

Then Открыть корзину

Then Перейти к оформлению заказа

Then Ввести имя "Даниил"

Then Ввести фамилию "Романов"

Then Ввести индекс "4978523"

Then Нажать на кнопку подтверждения

Then Подтвердить покупку

Then Видно элемент с текстом "FAIL"

Feature: Прохождение авторизации

Открытие страницы с авторизацией, логин

Scenario: Логин под существующим пользователем

Then Ввести логин "standard\_user"

Then Ввести пароль "secret\_sauce"

Then Нажать на кнопку подтверждения

Then Видно элемент с текстом "Products"

Scenario: Логин под несуществующим пользователем

Then Ввести логин "test"

Then Ввести пароль "test"

Then Нажать на кнопку подтверждения

Then Видно элемент с текстом "Epic sadface"

Feature: Добавление продукта в корзину

Добавить продукты в корзину, проверить, что добавились

Scenario: Добавление одного продукта в корзину

Given Пользователь залогинился

Then Добавить в корзину "Sauce Labs Backpack"

When Счётчик товаров равен "1"

Then Открыть корзину

When В корзине товар с именем "Sauce Labs Backpack"

Then Убрать элемент из корзины с именем "Sauce Labs Backpack"

When Корзина пуста

Scenario: Добавление нескольких продуктов в корзину

Given Пользователь залогинился

Then Добавить в корзину "Sauce Labs Backpack"

Then Добавить в корзину "Sauce Labs Bolt T-Shirt"

Then Добавить в корзину "Test.allTheThings() T-Shirt (Red)"

When Счётчик товаров равен "3"

Then Открыть корзину

When В корзине товар с именем "Sauce Labs Backpack"

When В корзине товар с именем "Sauce Labs Bolt T-Shirt"

When В корзине товар с именем "Test.allTheThings() T-Shirt (Red)"

Then Очистить корзину

When Корзина пуста

Scenario: Добавление всех продуктов в корзину

Given Пользователь залогинился

Then Добавить в корзину все продукты

When Счётчик товаров равен "6"

Then Открыть корзину

When В корзине "6" товаров

Then Очистить корзину

When Корзина пуста

Feature: Проверка соответсвия товара на странице с продуктами и в корзине

Добавить продукты в корзину, проверить что описание, название и цена одинаковые в корзине и на странице с продуктами

Scenario: Добавить один товар в корзину, проверить, что описание, цена и название одинаковы

Given Пользователь залогинился

When Добавить в корзину "Sauce Labs Backpack"

When Запомнить параметры "Sauce Labs Backpack" на главной странице

Then Открыть корзину

When В корзине "1" товаров

Then В корзине товар с именем "Sauce Labs Backpack"

When Запомнить параметры "Sauce Labs Backpack" в корзине

Then Проверить парамерты

Then Очистить корзину

Scenario: Добавить несколько товаров в корзину, проверить, что описание, цена и название одинаковы

Given Пользователь залогинился

When Добавить в корзину "Sauce Labs Backpack"

When Добавить в корзину "Sauce Labs Bolt T-Shirt"

When Запомнить параметры "Sauce Labs Backpack" на главной странице

When Запомнить параметры "Sauce Labs Bolt T-Shirt" на главной странице

Then Открыть корзину

When В корзине "2" товаров

Then В корзине товар с именем "Sauce Labs Backpack"

Then В корзине товар с именем "Sauce Labs Bolt T-Shirt"

When Запомнить параметры "Sauce Labs Backpack" в корзине

When Запомнить параметры "Sauce Labs Bolt T-Shirt" в корзине

Then Очистить корзину

# ПРИЛОЖЕНИЕ Б

package runners;

import io.cucumber.testng.AbstractTestNGCucumberTests;

import io.cucumber.testng.CucumberOptions;

import org.testng.annotations.Test;

@Test

@CucumberOptions(

plugin = {"pretty"},

tags = "",

features = "src/test/resources/buy",

glue = "steps")

public class BuyTestsRunner extends AbstractTestNGCucumberTests {

}

package runners;

import io.cucumber.testng.AbstractTestNGCucumberTests;

import io.cucumber.testng.CucumberOptions;

import org.testng.annotations.Test;

@Test

@CucumberOptions(

plugin = {"pretty"},

tags = "",

features = "src/test/resources/login",

glue = "steps")

public class LoginTestsRunner extends AbstractTestNGCucumberTests {

}

package runners;

import io.cucumber.testng.AbstractTestNGCucumberTests;

import io.cucumber.testng.CucumberOptions;

import org.testng.annotations.Test;

@Test

@CucumberOptions(

plugin = {"pretty"},

tags = "",

features = "src/test/resources/products",

glue = "steps")

public class ProductsTestsRunner extends AbstractTestNGCucumberTests {

}

# ПРИЛОЖЕНИЕ В

package pages;

import com.codeborne.selenide.ElementsCollection;

import com.codeborne.selenide.SelenideElement;

import static com.codeborne.selenide.Selenide.\*;

public class BasePage {

public ElementsCollection getNames() {

return $$(".inventory\_item\_name");

}

public ElementsCollection getPrices() {

return $$(".inventory\_item\_price");

}

public ElementsCollection getDescriptions() {

return $$(".inventory\_item\_desc");

}

public SelenideElement getDescription(String product) {

return $x(".//div[contains(text(),'" + product + "')]/following::div[@class='inventory\_item\_desc'][1]");

}

public SelenideElement getPrice(String product) {

return $x(".//div[contains(text(),'" + product + "')]/following::div[@class='inventory\_item\_price'][1]");

}

public ElementsCollection addButtons() {

return $$x(".//button[contains(text(),'Add to cart')]");

}

public ElementsCollection removeButtons() {

return $$x(".//button[contains(text(),'Remove')]");

}

public SelenideElement getButtonByProduct(String arg0) {

return $x(".//div[contains(text(),'" + arg0 + "')]/following::button[1]");

}

}

package pages.buy;

import com.codeborne.selenide.SelenideElement;

import static com.codeborne.selenide.Selenide.$;

public class CheckoutPage {

public SelenideElement firstName() {

return $("#first-name");

}

public SelenideElement lastName() {

return $("#last-name");

}

public SelenideElement postalCode() {

return $("#postal-code");

}

public SelenideElement finish() {

return $("#finish");

}

}

package pages.login;

import com.codeborne.selenide.SelenideElement;

import org.openqa.selenium.By;

import static com.codeborne.selenide.Selenide.$;

public class LoginPage {

public SelenideElement login() {

return $(By.cssSelector("#user-name")).as("Поле ввода логина");

}

public SelenideElement password() {

return $(By.cssSelector("#password")).as("Поле ввода пароля");

}

}

package pages.product;

import com.codeborne.selenide.SelenideElement;

import pages.BasePage;

import static com.codeborne.selenide.Selenide.\*;

public class CartPage extends BasePage {

public SelenideElement checkout() {

return $("#checkout");

}

public SelenideElement continueShopping() {

return $("#continue-shopping");

}

}

package pages.product;

import com.codeborne.selenide.SelenideElement;

import pages.BasePage;

import static com.codeborne.selenide.Selenide.\*;

public class ProductPage extends BasePage {

public SelenideElement cart() {

return $(".shopping\_cart\_link");

}

public SelenideElement cartCounter() {

return $(".shopping\_cart\_badge");

}

}

# ПРИЛОЖЕНИЕ Г

package steps;

import com.codeborne.selenide.Screenshots;

import com.codeborne.selenide.Selenide;

import io.cucumber.java.After;

import io.cucumber.java.Before;

import io.cucumber.java.Scenario;

import io.cucumber.java.en.Then;

import io.qameta.allure.Allure;

import org.openqa.selenium.By;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import java.io.\*;

import static com.codeborne.selenide.Condition.visible;

import static com.codeborne.selenide.Selenide.$;

import static com.codeborne.selenide.Selenide.open;

public class BaseSteps {

@Before

public void openUrl() {

open("https://www.saucedemo.com/");

}

@Then("Нажать на кнопку подтверждения {string}")

public void clickOnSubmitButton(String text) {

$(By.xpath("//\*[@type='submit' and @value='" + text + "']")).click();

}

@Then("Нажать на кнопку подтверждения")

public void clickOnSubmitButton() {

$(By.xpath("//\*[@type='submit']")).click();

}

@Then("Видно элемент с текстом {string}")

public void checkElementIsVisible(String text) {

$(By.xpath(".//\*[contains(text(),'" + text + "')]")).shouldBe(visible);

}

}

package steps.buy;

import io.cucumber.java.en.Then;

import pages.buy.CheckoutPage;

public class BuySteps {

CheckoutPage checkout = new CheckoutPage();

@Then("Ввести имя {string}")

public void enterName(String name) {

checkout.firstName().val(name);

}

@Then("Ввести фамилию {string}")

public void enterLastName(String lastName) {

checkout.lastName().val(lastName);

}

@Then("Ввести индекс {string}")

public void enterPostal(String postal) {

checkout.postalCode().val(postal);

}

@Then("Подтвердить покупку")

public void enterContinue() {

checkout.finish().click();

}

}

package steps.buy;

import com.codeborne.selenide.Condition;

import com.codeborne.selenide.SelenideElement;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

import model.Product;

import org.testng.Assert;

import pages.product.CartPage;

import pages.product.ProductPage;

import java.util.ArrayList;

import java.util.List;

public class ProductSteps {

ProductPage product = new ProductPage();

List<Product> excepted = new ArrayList<>();

List<Product> actual = new ArrayList<>();

@Then("Добавить в корзину {string}")

public void addToBucket(String item) {

product.getButtonByProduct(item).click();

}

@When("Счётчик товаров равен {string}")

public void cartCounterEqualsTo(String count) {

product.cartCounter().shouldBe(Condition.exactText(count));

}

@Then("Добавить в корзину все продукты")

public void addAllItems() {

product.addButtons().forEach(SelenideElement::click);

}

@When("Запомнить параметры {string} на главной странице")

public void initMainItem(String item) {

actual.add(Product.builder()

.name(item)

.price(product.getPrice(item).text())

.description(product.getDescription(item).text())

.build());

}

@When("Запомнить параметры {string} в корзине")

public void initCartItem(String item) {

excepted.add(Product.builder()

.name(item)

.price(product.getPrice(item).text())

.description(product.getDescription(item).text())

.build());

}

@Then("Проверить парамерты")

public void checkParams() {

Assert.assertEquals(actual, excepted);

}

}

package steps.cart;

import com.codeborne.selenide.CollectionCondition;

import com.codeborne.selenide.SelenideElement;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

import pages.product.CartPage;

import pages.product.ProductPage;

import static com.codeborne.selenide.CollectionCondition.empty;

import static com.codeborne.selenide.CollectionCondition.itemWithText;

public class CartSteps {

ProductPage product = new ProductPage();

CartPage cart = new CartPage();

@Then("Открыть корзину")

public void openBucket() {

product.cart().click();

}

@When("В корзине товар с именем {string}")

public void cartHave(String item) {

cart.getNames().shouldBe(itemWithText(item));

}

@Then("Убрать элемент из корзины с именем {string}")

public void removeItem(String item) {

cart.getButtonByProduct(item).click();

}

@When("Корзина пуста")

public void isEmpty() {

cart.getNames().shouldBe(empty);

}

@Then("Очистить корзину")

public void clear() {

cart.removeButtons().forEach(SelenideElement::click);

}

@When("В корзине {string} товаров")

public void haveOnlyItems(String countItems) {

cart.getNames().shouldBe(CollectionCondition.size(Integer.parseInt(countItems)));

}

@Then("Перейти к оформлению заказа")

public void checkoutOrder() {

cart.checkout().click();

}

}

package steps.login;

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import pages.login.LoginPage;

import steps.BaseSteps;

public class LoginSteps {

BaseSteps basePage = new BaseSteps();

LoginPage loginPage = new LoginPage();

@Then("Ввести логин {string}")

public void inputLogin(String login) {

loginPage.login().val(login);

}

@Then("Ввести пароль {string}")

public void inputPassword(String password) {

loginPage.password().val(password);

}

@Given("Пользователь залогинился")

public void userLogin() {

inputLogin("standard\_user");

inputPassword("secret\_sauce");

basePage.clickOnSubmitButton();

}

}