Федеральное государственное автономное образовательное учреждение

высшего образования

"Российский университет транспорта" (МИИТ)

Кафедра «Цифровые технологии управления транспортными процессами»

**ОТЧЁТ**

**ПО ТЕХНОЛОГИЧЕСКОЙ ПРАКТИКЕ**

«Создание структуры фрейморка для автоматизации тестирования в крупных компаниях»

**Тип практики:** ПРОИЗВОДСТВЕННАЯ ПРАКТИКА

(технологическая - 2)

Выполнил ст. г. УВА-411 Д. Н. Романов

Принял руководитель практики

доцент кафедры ЦТУТП Е.Я. Соймина

Москва  
2022

|  |
| --- |
| УТВЕРЖДАЮ  Профессор кафедры ЦТУТП, д.т.н.  \_\_\_\_\_\_\_\_\_\_\_ Э. К. Лецкий  « » 2022 год |

**ЗАДАНИЕ НА ПРАКТИКУ**

**(технологическая - 2)**

студента группы УВА-411 Романова Даниил Николаевича

**Наименование темы:**

Создание структуры фрейморка для автоматизации тестирования в крупных компаниях.

**Необходимо изучить:**

1. Паттерны проектирования фреймворков для автоматизированного тестирования.
2. Подходы к автоматизированному тестированию в крупных компаниях.
3. Алгоритмизация процессов обхода дерева.

**Необходимо самостоятельно выполнить:**

* + - 1. Анализ подходов к тестированию в крупных компаниях.
      2. Разработка архитектуры гибридного фреймворка.
      3. Разработка основных алгоритмов для работы фреймворка

**Основные требования к проделываемой работе:**

1. Использовать ГОСТ Р 7.32-2001 – для оформления пояснительной записки.

**Источники информации:**

1. Роман Савин «Тестирование Дот Ком». – Москва: «Roman Savenkov», 2017. – 312 с.
2. Джейсон Арбон, Джефф Каролло, Уиттакер Джеймс «Как тестируют в Google» – СПб.:«Питер», 2014 – 320. с.

**Материалы к защите:**

1. Заполненный, подписанный и утверждённый лист индивидуального задания на технологическую практику.
2. Пояснительная записка (отчёт).

|  |  |
| --- | --- |
| **Студент группы** **УВА-411** | Романов Д.Н. |
| **Руководитель практики** | Соймина Е.Я. |

Оглавление

[1. Введение 4](#_Toc105265849)

[2. Тестирование как элемент цикла разработки ПО 5](#_Toc105265850)

[3. Разработка архитектуры фреймворка 8](#_Toc105265851)

[4. Модуль с тестовыми сценариями 10](#_Toc105265852)

[5. Связующий модуль 11](#_Toc105265853)

[6. Функциональный модуль 12](#_Toc105265854)

[7. Алгоритмы необходимые для реализации модулей фрейворка. 14](#_Toc105265855)

[8. Заключение 18](#_Toc105265856)

### 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 практики:

* анализ подходов к тестированию в крупных компаниях;
* разработка архитектуры гибридного фреймворка;
* разработка основных алгоритмов для работы фреймворка.

### Тестирование как элемент цикла разработки ПО

Современный цикл разработки программного обеспечения совмещает в себе несколько этапов - анализ, разработка, тестирование и релиз [3].

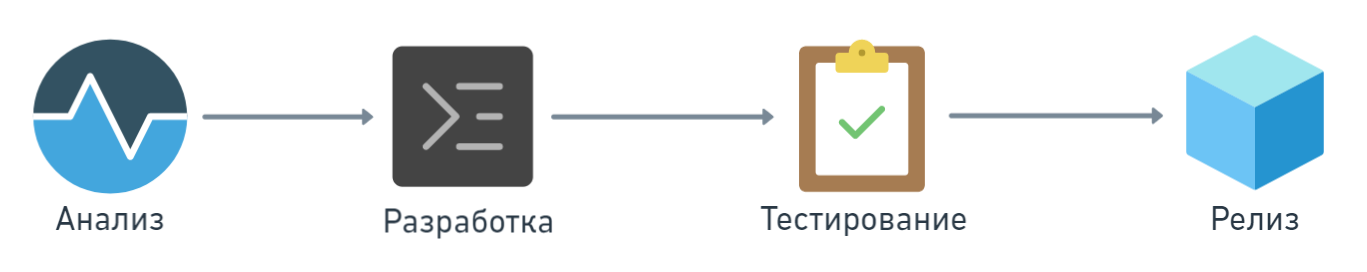


Рис. 1 - Современный цикл разработки ПО

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

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.

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.

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



Рис. 2 - 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.

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.

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.

Итого мы имеем 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.

### Разработка архитектуры фреймворка

Предлагается разработать приложение из трёх компонентов:

* Функциональный модуль. Содержит основные функциональные возможности автоматизированного теста.
* Модуль с тестовыми сценариями. Содержит тестовые сценарии.
* Связующий модуль. Преобразует тестовые сценарии в полноценный автоматизированный тест.

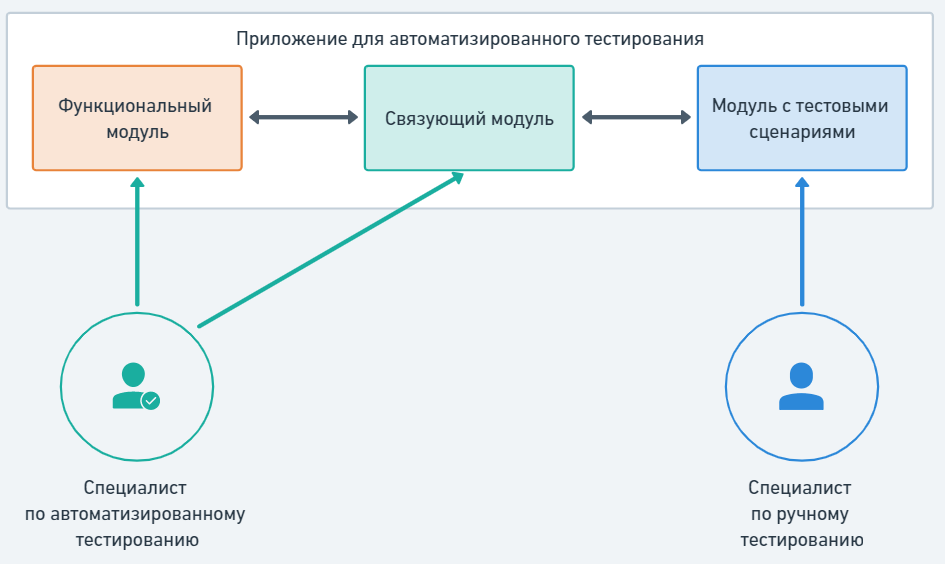


Рис. 3 - Схема взаимодействия тестировщика и автотестера №1

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.

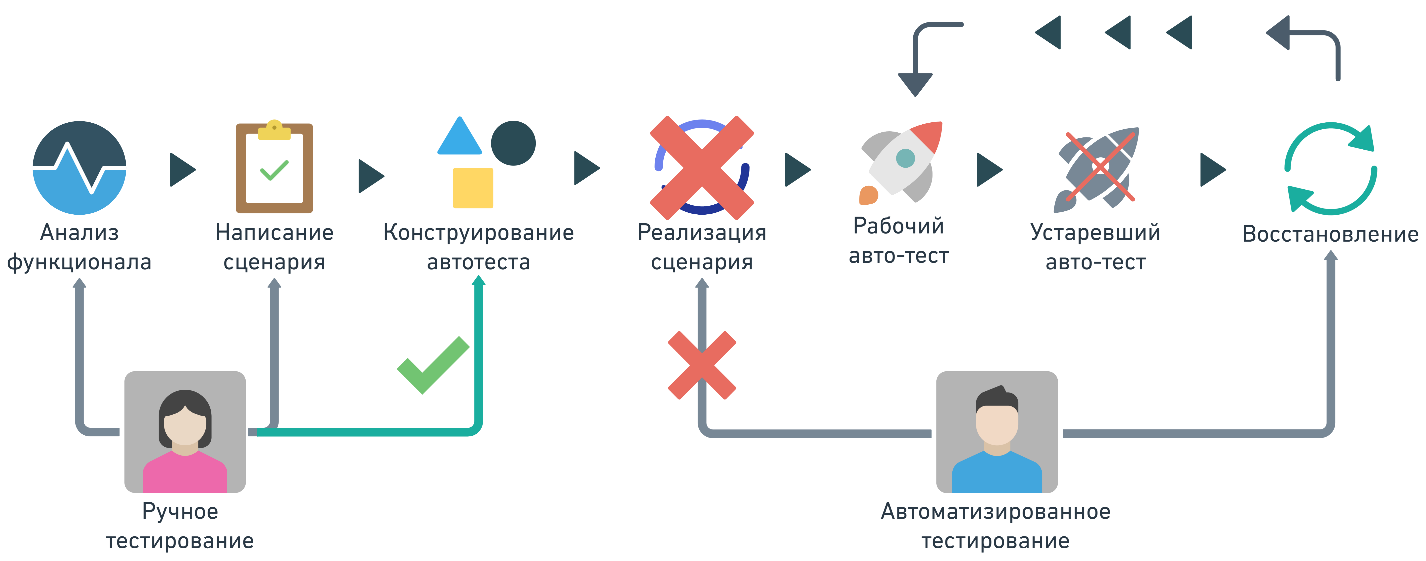


Рис. 4 - Схема взаимодействия тестировщика и автотестера №2

Такой вариант решения проблемы тестирования ПО позволит специалистам по ручному тестированию функционально конструировать автоматизированные тесты.

Целевое решение имеет ряд преимуществ, таких как:

* Тестовые сценарии, написанные на русском языке
* Высокая скорость тестирования.
* Низкая стоимость тестирования.
* Повышение эффективности работы специалистов по ручному тестированию.
* Повышение эффективности работы специалистов по автоматизации тестирования.
* Конструктор для создания автоматизированных тестов.

### Модуль с тестовыми сценариями

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.

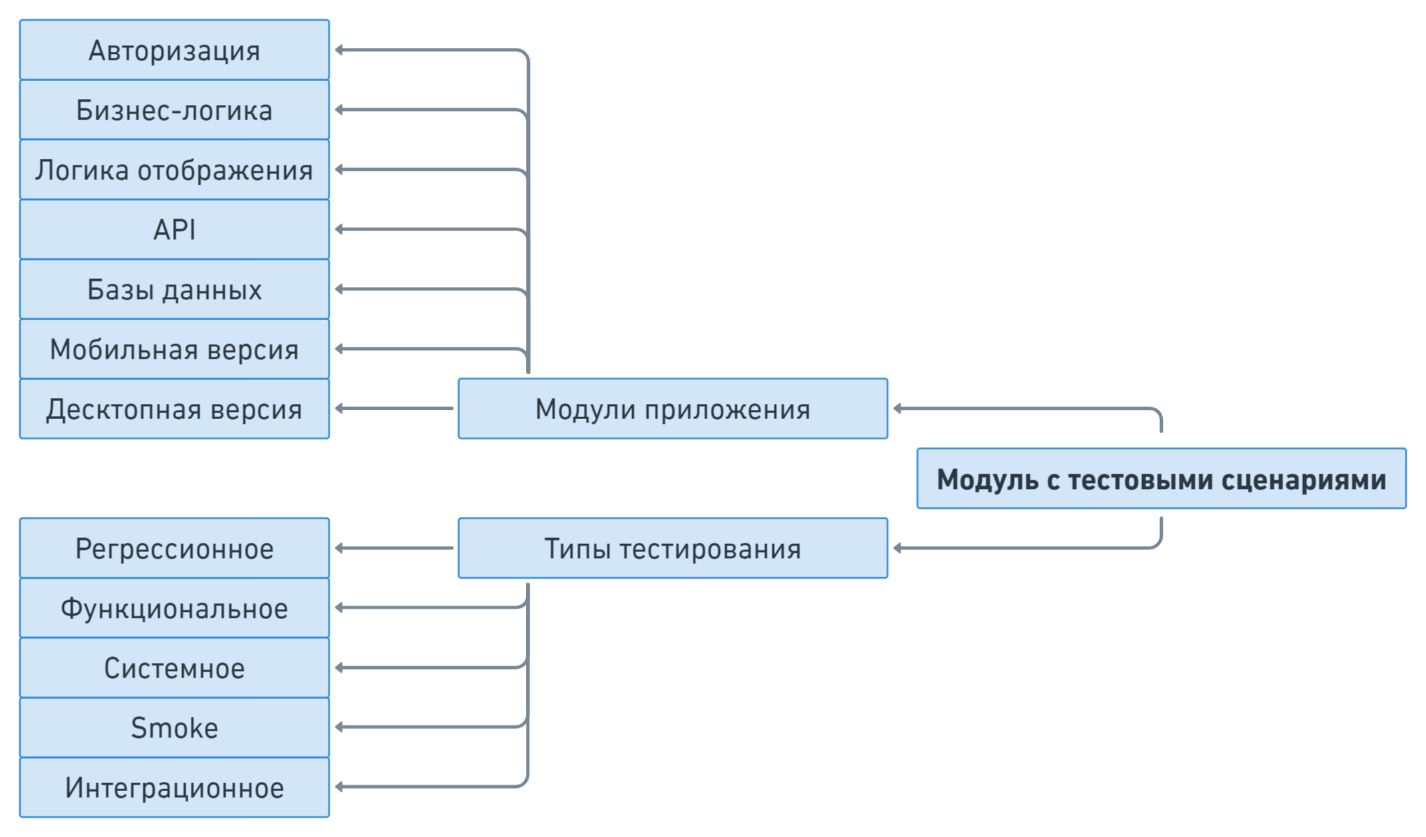


Рис. 5 – Структура сценариев для интернет-магазина

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

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

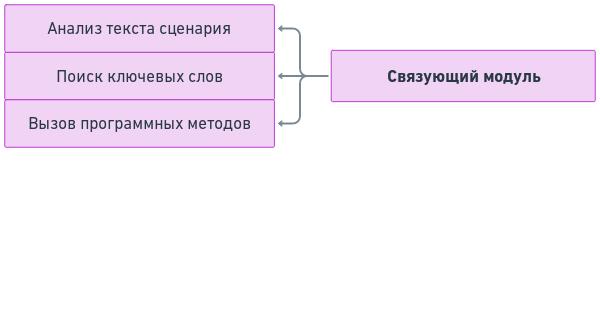


Рис. 6 - Структурная схема связующего модуля

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.

### Функциональный модуль

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 структуру функционального модуля можно представить следующим образом.

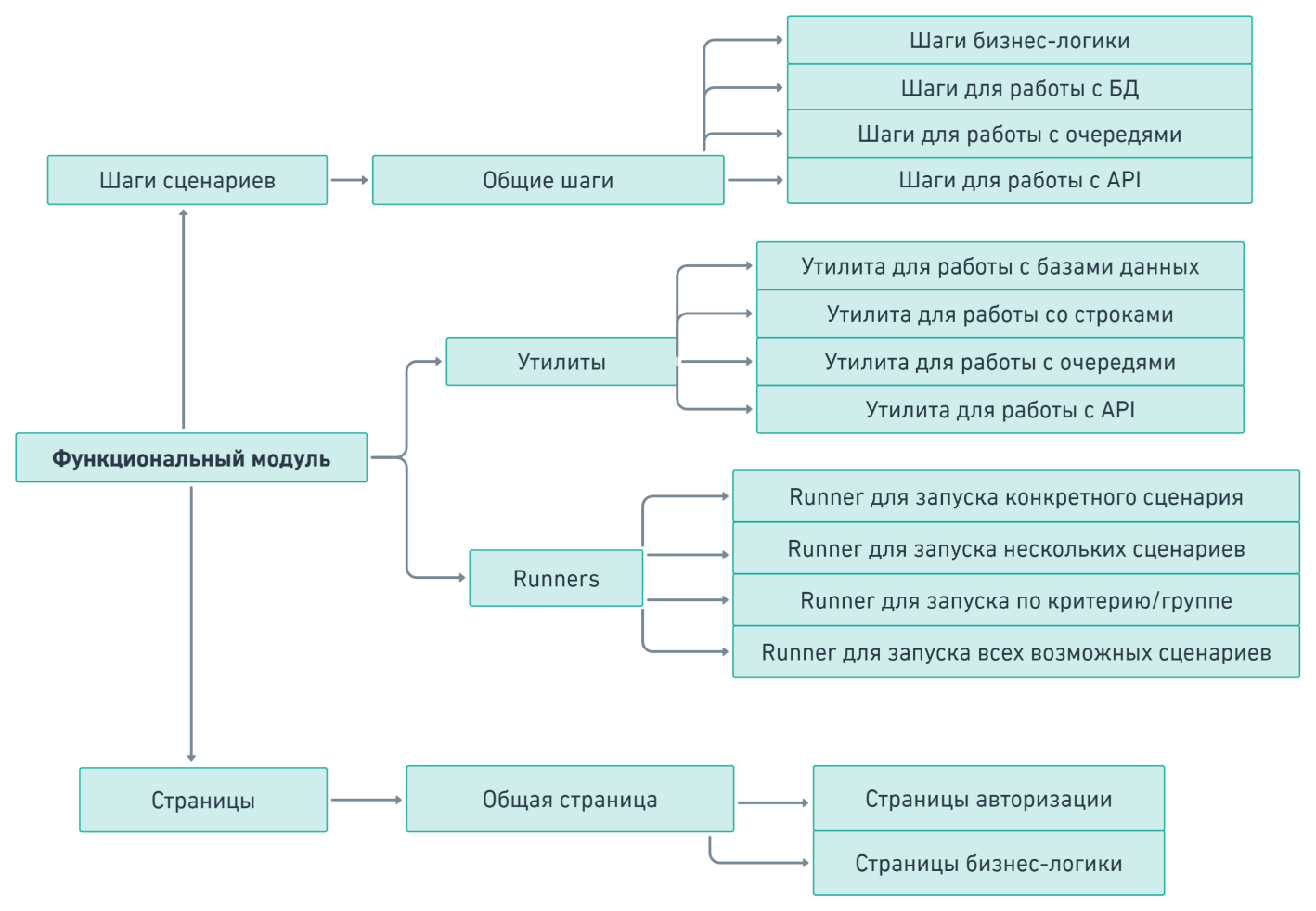


Рис. 8 – Структура классов и директорий

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.

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

Функциональный модуль в данной реализации представляет собой полноценный интерфейс для управления браузером, БД, API, который способен эмулировать поведения пользователя в любых комбинациях. Есть возможность отрывать браузер, нажимать на кнопки, искать элементы на странице, перетаскивать их, скачивать и обрабатывать файлы.

### Алгоритмы необходимые для реализации модулей фрейворка.

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 реализован следующим образом.



Рис. 9 – 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.

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.



Рис. 10 – Алгоритм написания тестового сценария

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

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.

### Заключение

В ходе практики построена архитектура гибридного фреймворка для написания автоматизированных тесто, были проанализированы основные модули, составлена общая схема взаимодействия внутри фреймворка, разработаны алгоритмы работы внутри функционального модуля и модуля тестовых сценариев.

**Источники информации**

1. Основы тестирования программного обеспечения. Учебное пособие для СПО, 2-е изд., стер.
2. Software Testing Techniques, 2nd Edition, Борис Бейзер
3. Scrum: Революционный метод управления проектами, Джефф Сазерленд
4. Experiences of Test Automation: Case Studies of Software Test Automation 1st Edition, Dorothy Graham