Министерство образования и науки Российской Федерации

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Математики, физики и информационных технологий
(институт)
Прикладная математика и информатика
(кафедра)
09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)
Бизнес-информатика
(наименование профиля, специализации)

БАКАЛАВРСКАЯ РАБОТА

на тему **Разработка фреймворка автоматизированного тестирования веб-приложений**

Студент А.В. Жилин _____
(И.О. Фамилия) (личная подпись)

Руководитель А.В. Очеповский _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой канд.тех.наук, доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____20_____г.

Тольятти 2018

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Аннотация

Тема бакалаврской работы – «Разработка фреймворка автоматизированного тестирования веб-приложений».

Цель работы: разработка универсального фреймворка автоматизированного тестирования веб-приложений.

Объект исследования: обеспечение качества и надежности программного обеспечения.

Предмет исследования: инструмент – фреймворк для тестирования, служащий для обеспечения удобства при разработке автоматизированных тестов для широкого круга специалистов отделов/управлений тестирования.

Работа посвящена исследованию важной области в современной информатике – автоматизированное тестирование программного обеспечения. Данный вид деятельности активно применяется в процессе жизненного цикла разработки ПО и непосредственно влияет на обеспечение качества ПО. Поэтому изучение проблем автоматизации тестирования и разработка инструментария для его проведения является актуальной задачей.

В первой главе проведен анализ деятельности специалистов по ручному тестированию, описаны предпосылки для автоматизации и представлено обоснование необходимости разработки инструмента (фреймворка) для написания автоматизированных тестов.

Во второй главе разработаны логическая и функциональная схемы проекта фреймворка, представлены данные о разработке, а также контрольный пример реализации проекта.

В третьей главе представлено экономическое обоснование эффективности проекта на основе расчетов себестоимости и показателей итоговой эффективности проекта.

Бакалаврская работа состоит из ведения, трех глав, заключения, списка используемых источников и приложений. Пояснительная записка выполнена на

72 страницах, включает 26 рисунков, 5 таблиц. Список используемых источников включает 14 источников, из них 4 на иностранном языке.

Оглавление

Введение

Контроль качества ПО — «это планомерная и систематичная программа действий, призванная гарантировать, что система обладает желательными характеристиками»[6].

Тестирование ПО является важной частью процесса контроля качества, так как позволяет проверить и оценить соответствие разработанного ПО требованиям к его функциональности.

Цель автоматизации тестирования – автоматическое выполнение тестов вместо ручного труда специалистов по тестированию. Автоматизация тестирования является актуальной задачей для компаний в сфере разработки программного обеспечения, так как позволяет сократить как время на проведение тестирования, так и количество необходимых для проведения тестирования специалистов, что, несомненно, способствует увеличению экономической эффективности IT-компаний.

При этом проблемой является вовлечение специалистов по ручному тестированию в новую для себя сферу деятельности – обеспечение работы автоматических тестов. Для упрощения данного процесса могут быть разработаны специальные инструменты – фреймворки автоматизированного тестирования, дающие возможность сосредоточиться на конкретной задаче, без разработки вспомогательного и служебного инструментария. Также проблемой является трудоемкость ручного тестирования.

Актуальность решения данных проблем является следствием развития использования гибких методологий разработки программного обеспечения, основной задачей которых является оптимизация производственного процесса и минимизация рисков путем сведения разработки к серии коротких циклов.

Практическая значимость работы заключается в том, что разрабатываемый фреймворк может быть использован широким кругом как индивидуальных разработчиков программного обеспечения, так и корпоративными пользователями, такими как: отделы/управления тестирования

в рамках организационных структур (отделов, управлений) разработки программного обеспечения.

Объектом исследования является обеспечение качества и надежности программного обеспечения.

Предметом исследования является инструмент – фреймворк для тестирования, служащий для обеспечения удобства при разработке автоматизированных тестов для широкого круга специалистов отделов/управлений тестирования.

Целью данной выпускной квалификационной работы является разработка универсального фреймворка для автоматизации тестирования веб-приложений.

Задачи (этапы) для достижения цели: обозначить предпосылки для автоматизации тестирования и проблемы, возникающие при этом, построить проект и разработать фреймворк для автоматизации тестирования, провести анализ экономической эффективности проекта.

В первой главе проведен анализ деятельности специалистов по ручному тестированию, описаны предпосылки для автоматизации и представлено обоснование необходимости разработки инструмента (фреймворка) для написания автоматизированных тестов.

Во второй главе разработаны логическая и функциональная схемы проекта фреймворка, представлены данные о разработке, а также контрольный пример реализации проекта.

Основной проблемой при разработке средств автоматизации тестирования, является соединение в единую систему различных компонентов, отвечающих за автоматизацию, таких как: драйверы для работы с GUI-интерфейсом веб-приложений, драйверы для работы с API-интерфейсами, средства формирования, генерации и хранения тестовых данных, средства по работе с логами, с печатными формами, средства для создания отчетов по тестированию.

Также имеют место при рассмотрении задачи создания единого фреймворка автоматизированного тестирования особенности конкретных

тестируемых веб-приложений, такие как: работа с конкретными браузерами (версиями браузеров), работа под конкретными операционными системами, работа со специфическими подключаемыми устройствами. Таким образом, при разработке потребуется продумать и спроектировать максимально универсальную систему, не учитывающую частные случаи, а обеспечивающую максимально удобную возможность расширения под конкретные системы и требования.

В третьей главе представлено экономическое обоснование эффективности проекта на основе расчетов себестоимости и показателей итоговой эффективности проекта.

Апробация решения производится на реальных задачах тестирования веб-приложений банковской сферы, что позволяет оценить удобство и практическую применяемость разрабатываемого решения.

Глава 1. Анализ проблемы автоматизированного тестирования веб-приложений

1.1 Технико-экономическая характеристика автоматизированного тестирования веб-приложений

1.1.1. Характеристика процессов разработки ПО

В соответствии со стандартом ИСО/МЭК 12207-2010 [4] процессы жизненного цикла программных средств включают в себя следующие этапы (рассматриваются этапы, касающиеся процессов реализации и технических процессов):

1. Определение требований.
2. Анализ требований.
3. Проектирование архитектуры.
4. Детальное проектирование.
5. Конструирование программного средства.
6. Комплексирование программных средств.
7. Квалификационное тестирование.
8. Инсталляция.
9. Приемка.
10. Сопровождение.
11. Прекращение применения программного средства.

Данная работа касается рассмотрения этапов квалификационного тестирования (7 пункт), а также приемки программных средств (9 пункт).

«Цель процесса квалификационного тестирования системы заключается в подтверждении того, что реализация каждого системного требования тестируется на соответствие и система готова к поставке» [4].

«Цель процесса поддержки приемки программных средств заключается в содействии приобретающей стороне в обеспечении уверенности в том, что продукт соответствует заданным требованиям» [4]. Таким образом, оба этапа –

и квалификационное тестирование и приемка служат схожим целям – проверка готовности ПО и соответствия его требованиям.

В соответствии со стандартом IEEE SWEBOK v3.0 «тестирование программного обеспечения (Software Testing) – это динамическая проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом» [12]. Тестирование является одной из составляющих процесса контроля качества и включает в себя:

- планирование процесса тестирования (Test Management);
- разработку и проектирование тестов (Test Design);
- прогон тестовых сценариев (Test Execution);
- анализ результатов, полученных после прогонов тестов (Test Analysis).

В соответствии с пособием по стандартизации и сертификации программного обеспечения «все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы: 1) функциональные; 2) нефункциональные; 3) связанные с изменениями» [7].

«Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены самые распространенные виды функциональных тестов:

- Тестирование безопасности (Security and Access Control Testing);
- Тестирование взаимодействия (Interoperability Testing)» [7].

«Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, "Как"

система работает. Далее перечислены основные виды нефункциональных тестов:

- Все виды тестирования производительности:

а) нагрузочное тестирование (Performance and Load Testing);

б) стрессовое тестирование (Stress Testing);

в) тестирование стабильности или надежности (Stability / Reliability Testing);

г) объемное тестирование (Volume Testing);

д) тестирование удобства пользования (Usability Testing);

е) тестирование на отказ и восстановление (Failover and Recovery Testing);

ж) конфигурационное тестирование (Configuration Testing)» [7].

1.1.2. Проблема автоматизации тестирования

В соответствии с информацией портала «Про Тестинг» [8] «Автоматизированное тестирование программного обеспечения (Software Automation Testing) - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

Специалист по автоматизированному тестированию программного обеспечения (Software Automation Tester) - это технический специалист (тестировщик или разработчик программного обеспечения), обеспечивающий создание, отладку и поддержку работоспособного состояния тестовых скриптов, тестовых наборов и инструментов для автоматизированного тестирования.

Инструмент для автоматизированного тестирования (Automation Test Tool) - это программное обеспечение, посредством которого специалист по автоматизированному тестированию осуществляет создание, отладку, выполнение и анализ результатов прогона тестовых скриптов.

Тестовый Скрипт (Test Script) - это набор инструкций, для автоматической проверки определенной части программного обеспечения.

Тестовый набор (Test Suite) - это комбинация тестовых скриптов, для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

Тесты для запуска (Test Run) - это комбинация тестовых скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования)» [8].

Для написания и работы данных артефактов удобнее всего применять общий фреймворк автоматизированного тестирования.

Основной проблемой при автоматизации является подготовка и контроль перечисленных артефактов, а также настройка необходимой среды для запуска и выполнения автоматических тестов. При этом есть проблема вовлечения в процесс поддержки автоматических тестов как можно большего количества специалистов, занимающихся тестированием, связанная с возможным недостатком квалификации или специализированных знаний языков и сред разработки.

Для написания автоматических тестов, большинство инструментов автоматизации требуют от тестировщика понимания скриптового языка (VB Script, Java Script, и т.д.). Обычно инструменты позволяют создавать тесты при помощи записи и воспроизведения, но, как правило, такие скрипты не очень эффективны, не могут быть переиспользованы и тяжелы в поддержке. Фреймворк автоматического тестирования - это набор условий, концепций и практик, направленный на переиспользование, уменьшение затрат на поддержку и повышение надежности использования тестов.

Использование фреймворка решает большинство описанных проблем, что предполагает использование инструмента широким кругом специалистов, включая разработчиков и специалистов по ручному тестированию.

1.2. Концептуальное моделирование предметной области

В результате анализа деятельности по тестированию программного обеспечения и была разработана модель IDEF0 «Как есть». Существующая технология тестирования предполагает ручное тестирование, как нового функционала, так и смоук и регрессионных тестов, что требует значительных человеческих ресурсов. Схема IDEF0 деятельности отдела тестирования ПО и декомпозиция текущих этапов/ операций деятельности отдела тестирования DFD представлена на рисунках 1.1 и 1.2.

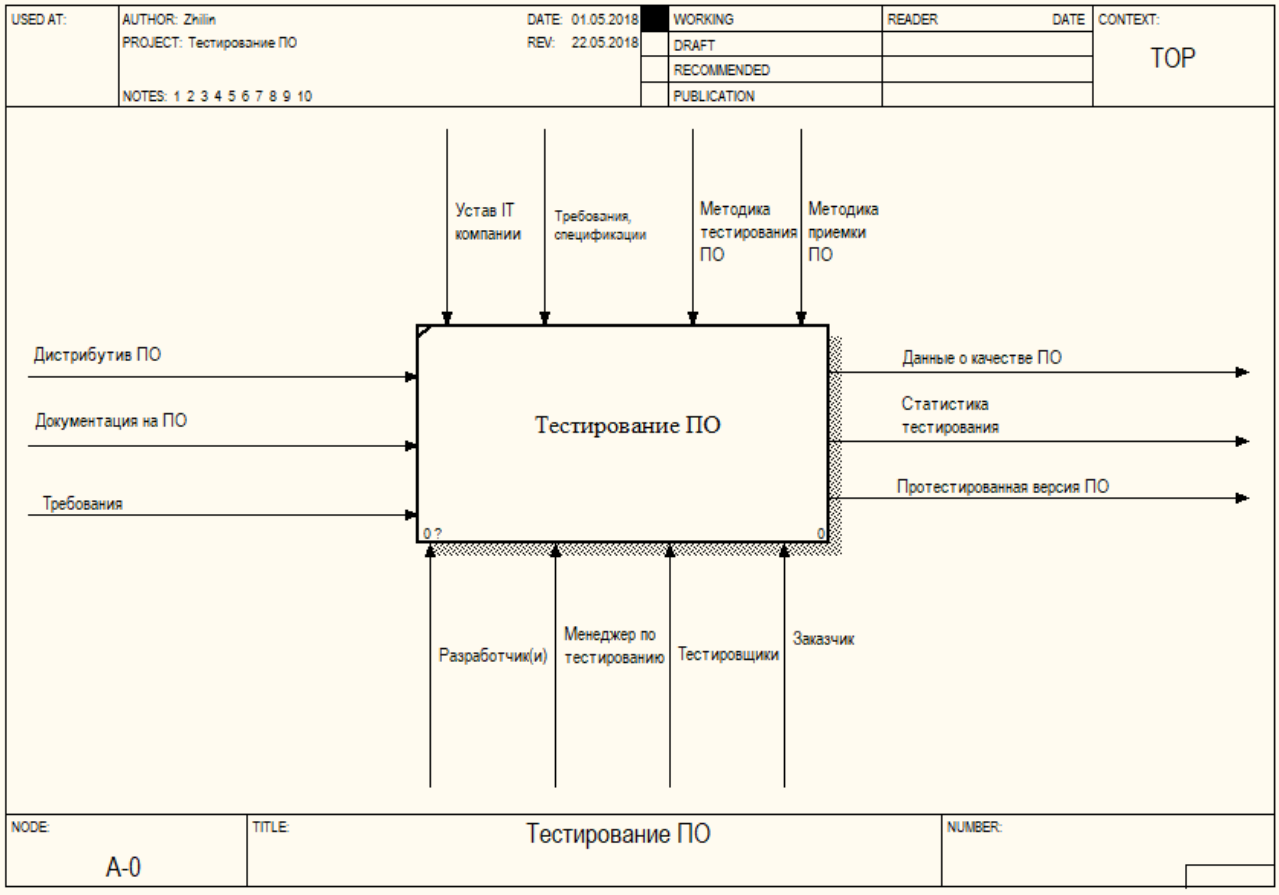

Рисунок 1.1. Схема процесса работы отдела тестирования «Как есть»

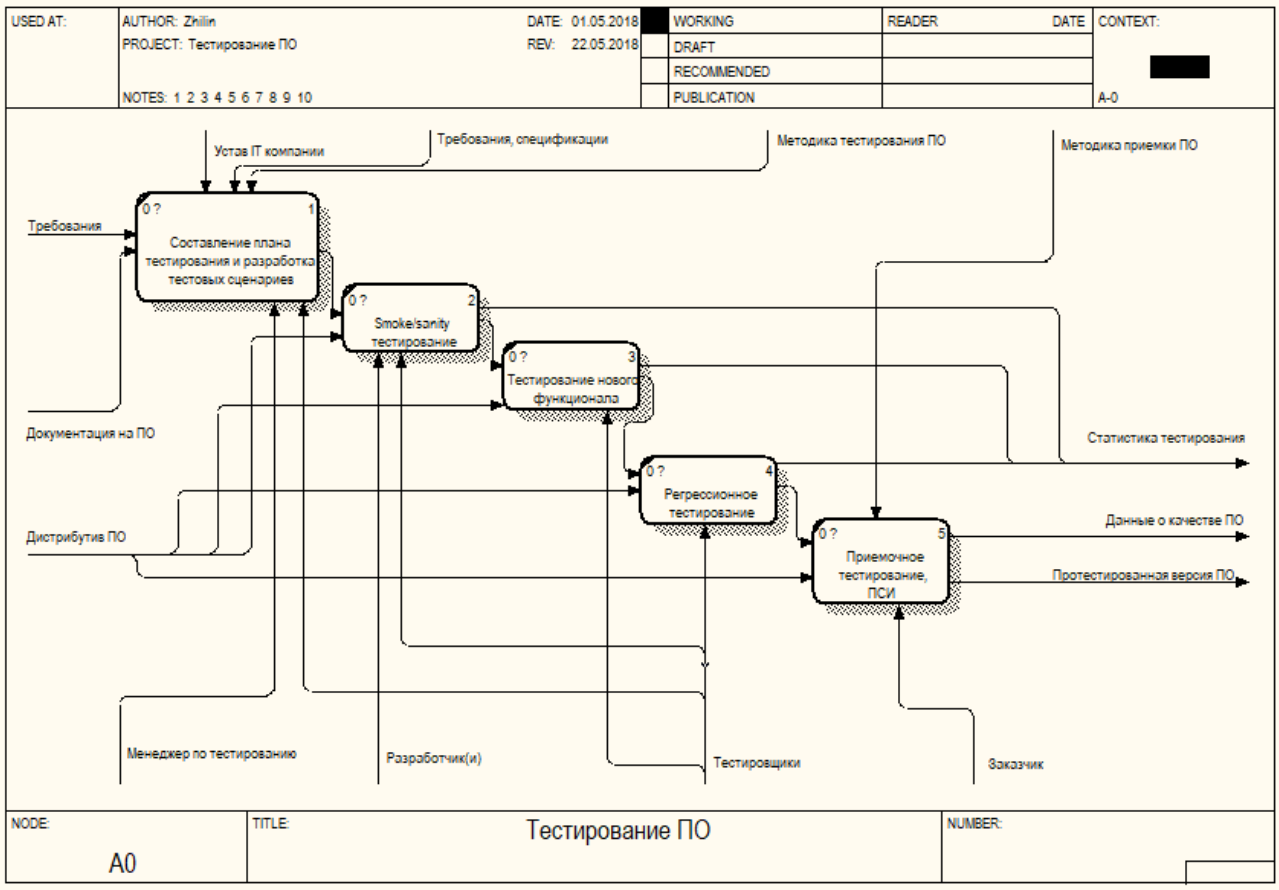

Рисунок 1.2. Декомпозиция схемы процесса работы отдела тестирования «Как есть»

Выявленные недостатки по данной схеме процесса:

- низкая производительность труда специалистов по тестированию;

- низкое качество отчетной информации из-за возможных ошибок специалистов при ручном отражении результатов прогонов тестов;

- высокая трудоемкость тестирования крупных релизов;

- несовершенство организации первичных тестовых данных разными сотрудниками;

- несовершенство процесса анализа возникающих при тестировании проблем и ошибок;

- разработчикам приходится принимать участие в процессе тестирования на этапе смоук тестирования, так как установка поставляемого ПО проводится в ручном режиме без использования методики Continuous Integration/Delivery.

Заказчик представлен на схеме в качестве участника приемочного тестирования продукта. Проблема увеличения эффективности приемочного тестирования также может рассматриваться в рамках проблематики данной работы.

1.3. Постановка задачи разработки фреймворка автоматизированного тестирования

1.3.1. Решение проблем автоматизации тестирования веб-приложений

Решение описанных в предыдущем параграфе проблем, предлагается осуществить с помощью разработки и внедрения следующей функциональности. Ключевые функции разрабатываемого фреймворка автоматизации тестирования:

1. Keyword Driven подход

Если тестировщики не являются экспертами скриптовых языков, следует иметь возможность заменить скрипты тестами на основе ключевых слов (keyword-driven). Этот подход позволяет тестировщику (автоматизатору) создавать автоматические тесты просто описывая каждый шаг теста. Например, при тестировании процесса авторизации, где пользователь должен запустить приложение, ввести свое имя пользователя и пароль, а затем нажать кнопку входа при обычном подходе тестировщикам нужно писать скрипт на выбранном языке программирования, например, Java, Python, VB Script, который бы запустил приложение, распознал каждый объект на экране (поля ввода имени, пароля, кнопку входа), далее вводил имя пользователя, пароль, и нажимал бы кнопку входа. При использовании подхода Keyword Driven, тестировщику не обязательно понимать скриптовый язык для того чтобы создать такой тест, ему достаточно описать эти события (запуск приложения, ввод имени пользователя "abc", ввод пароля "xxx", нажатие кнопки "вход") языком, максимально приближенном к естественному. Данный подход автоматизации упрощает работу специалистов по тестированию, чем написание только скриптов.

2. Итерации наборов данных

Данная практика позволяет тестировщикам использовать одни и те же автоматические тесты, но при этом запускать их с разными наборами тестовых данных. Например, можно использовать один тест для входа в систему, но выполнять его с различными комбинациями имени пользователя и пароля, чтобы протестировать несколько сценариев. Если фреймворк достаточно гибок и позволяет определение нескольких итераций для запуска с различными данными, то это уменьшит время, затрачиваемое на автоматизацию.

3. Распределенный запуск на нескольких компьютерах

Полезной возможностью является возможность распределенного запуска на нескольких компьютерах. Например, это полезно, если у компании есть потребность в создании лаборатории QA с несколькими серверами. При этом возникает потребность запуска тестов на нескольких серверах.

4. Автоматический запуск по времени

Для удобства контроля и планирования запусков следует предусмотреть возможность запускать их автоматически в назначенное время. Фреймворк должен иметь возможность автоматического запуска в заданное время (по дням недели), иметь возможность повторов (например, запускать каждый день в 6 часов). Данный функционал может быть реализован также на базе системы Continuous Integration, например, Jenkins/Hudson.

5. Отчеты результатов запуска

Во время разработки фреймворка необходимо продумать механизм удобных и понятных пользователям отчетов по прогонам тестовых сценариев. В этом случае, возможна потребность осуществления связи отчетности, которую выдает фреймворк, с уже существующим инструментом управления тестированием для ручного тестирования. При этом следует учесть возможность интеграции результатов автоматических тестов с отчетами инструментов тест менеджмента. Таким образом, будет обеспечена возможность обобщенно анализировать результаты активностей по ручному и автоматическому тестированию.

1.3.2. Общая характеристика организации решения задачи на ЭВМ

Поставленная задача будет решаться путем разработки фреймворка автоматизированного тестирования на языке программирования Java с использованием следующих инструментов разработки: IntelliJ IDEA, Maven, Git, Gitlab, Jenkins.

Изменение функций в подразделениях тестирования ПО после внедрения разработанного проекта будет заключаться в вовлечении большего количества сотрудников в разработку и использование автоматических тестов.

Источниками информации для разработки новых тестов будут являться разработанные сценарии ручного тестирования, а также имеющиеся спецификации и требования к тестируемому ПО.

Задача разработки и внедрения фреймворка будет решаться поэтапно путем внедрения в процесс тестирования автоматизированных тестов. При этом чтобы кардинально не менять существующий процесс тестирования, что влечет за собой появление дополнительных рисков для бизнеса, можно выбрать путь внедрения фреймворка только для новых разрабатываемых тестов. После первичной наладки, можно будет приступать к переносу уже имеющихся автотестов, для обеспечения унификации тестовой модели.

Этапами разработки будут:

1. Оценка потребности в первичном и дополнительном функционале фреймворка для пользователей.
2. Проектирование разработки.
3. Разработка/кодирование.
4. Модульное тестирование фреймворка.
5. Интеграционное тестирование фреймворка.
6. Внедрение.
7. Поддержка и дальнейшее развитие функционала фреймворка.

Для вывода результатов выполнения тестов выбраны следующие инструменты:

1. Для вывода консольных логов для анализа причин возможных причин падений тестов из-за их некорректной реализации будет использован инструмент SLF4J – Simple Logging Facade for Java. Данный инструмент является удобным фасадом с широкими возможностями настройки вывода для таких фреймворков логирования как java.util.logging, logback и log4j.

2. Для вывода отчетов в удобной для пользователей форме будет использован фреймворк Allure 2. Данный фреймворк разрабатывался изначально компанией Яндекс, и на текущий момент является свободным ПО с широкими возможностями настройки и привязки практически ко всем доступным инструментам unit-тестов для языка Java: jUnit4, jUnit5, TestNG, Cucumber JVM, Selenide.

1.4. Анализ существующих разработок в области тестирования веб-приложений

1.4.1. Определение критериев анализа

В качестве критериев для анализа рынка существующего ПО для автоматизации тестирования будут использоваться:

1. Распространенность инструмента – подразумевается наличие базы пользователей, за счет которых увеличивается вероятность нахождения ответов на возникающие вопросы по использованию инструмента, а также вероятность актуальности знаний для разработчиков. Важность данного критерия в том, чтобы разработчики автотестов имели возможность воспользоваться накопленным опытом других разработчиков, а не решали все проблемы «с нуля».

2. Поддержка – поддерживается ли фреймворк, то есть как часто выходят обновления/исправления ошибок. Поддержка важна для возможности исправления выявляемых ошибок, влияющих на стабильность работы системы.

3. Цена инструмента – вариантами будут платное ПО, бесплатное, условно бесплатное. Цена инструмента является экономическим критерием,

определяющим расчет прямой эффективности от внедрения программного средства для автоматизации.

4. Открытость исходного кода – открытый исходный код, либо проприетарный формат программного средства. Данный критерий позволяет оценить возможность доработки инструмента для собственных нужд.

5. Поддержка браузеров – так как рассматривается фреймворк именно для тестирования веб-приложений, важную роль играет поддержка новых версий популярных браузеров.

6. Поддержка операционных систем. Поддержка как можно большего количества операционных систем влияет на выбор того или иного инструмента, в том случае, если тестируемое ПО предполагает мультиплатформенность.

7. Язык программирования – язык программирования, на котором написан фреймворк влияет для бизнеса на простоту/сложность поиска специалистов для разработки и поддержки тестов и самого фреймворка.

8. Возможности взаимодействия со сторонними библиотеками в рамках Continious Integration/Delivery. От данного критерия зависит, возможно ли встроить тесты в процесс непрерывной разработки и поставки ПО, что является важным критерием для оценки применимости инструмента.

9. Возможности работы с моделью BDD (Behaviour Driven Development). Данный критерий отвечает за решение недостатка необходимости наличия специализированных знаний по разработке ПО у пользователей фреймворка.

10. Простота установки, настройки и удобность пользования. Аналогично пункту 9 данного списка, простота использования будет влиять на расширение списка пользователей фреймворка.

1.4.2. Сравнительная характеристика существующих разработок

В данном разделе представлено сравнение разрабатываемого фреймворка с существующими на рынке распространенными решениями в данной области по описанным в предыдущем пункте критериям.

Таблица 1.1. Сравнение фреймворков автоматизации тестирования

Категория/Продукт	Разрабатываемый фреймворк	Selenium	Katalon	UFT	Test Complete	Watir
Распространенность	-	+++	++	+	++	++
Поддержка инструмента	+++	+++	+	+	++	+
Цена	Бесплатный / внутренние затраты	Бесплатный	Бесплатный	Дорогой	Средняя цена	Бесплатный
Открытость	+++	+++	-	-	-	+++
Поддержка браузеров	IE, Chrome, Firefox, Opera, Edge, мобильные браузеры	IE, Chrome, Firefox, Opera, Edge, мобильные браузеры	IE, Chrome, Firefox	IE, Chrome, Firefox, Safari, мобильные браузеры	IE, Chrome, Firefox, Opera, Safari, Edge	IE, Chrome, Firefox, Safari, Edge
Категория/Продукт	Разрабатываемый фреймворк	Selenium	Katalon	UFT	Test Complete	Watir
Поддержка ОС	Windows, Linux OS X	Windows, Linux OS X	Windows, Linux OS X	Windows	Windows	Windows, Linux OS X
Язык программирования	Java	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, Jscript, Delphi, C++, C#	Ruby
CI/CD	+++	+++	++	+++	+++	+
BDD	+++	-	-	-	++	+++
Простота и удобность	+++	-	++	-	+	-

Условные обозначения:

+/++/+++ - соответствие категории/критерию. Чем больше плюсов, тем лучше.

- - несоответствие критерию/отсутствие функциональности.

Основными достоинствами разрабатываемого фреймворка, по сравнению с имеющимися на рынке продуктами, будут являться:

1. Условная бесплатность. Затратами будут только начальные затраты на разработку, а также затраты на поддержку и разработку нового функционала. Плюс данного подхода в том, что на разработку будут использованы внутренние ресурсы. Таким образом, будет реализован только необходимый функционал, и будут отсутствовать установленные периодические обязательные платежи.

2. Поддержка инструмента. Так как поддержка будет осуществляться внутренним персоналом компании, это будет обеспечивать необходимую оперативность в решении проблем и в осуществлении необходимых доработок.

3. Открытость. Планируется реализация проекта с исходными кодами на принципах свободного программного обеспечения. Таким образом, появляется шанс на получение разрабатываемым инструментом широкого распространения. Таким образом, можно получить бонус как при поддержке продукта, так как будут привлечены сторонние разработчики, желающие внести свой вклад в открытый проект, так и бонус к репутации компании разработчика данного фреймворка.

4. Язык программирования и поддержка ОС. Языком разработки фреймворка выбран язык Java, как наиболее распространенный на сегодня по ряду индексов. Например, в соответствии с широко известным индексом TIOBE Programming Community Index на май 2018 года, язык Java занимает первое место по распространенности с рейтингом 16,38% [14]. Таким образом, решается проблема со сложностью потенциального поиска новых разработчиков на проект. Аналогично, это влияет на потенциальное количество разработчиков/тестировщиков, которые будут являться пользователями фреймворка.

5. Поддержка браузеров. Фреймворк поддерживает практически все имеющиеся на рынке браузеры, так как основан на основе Selenium Webdriver, который поддерживает как драйверы собственной разработки, так и разработанные любыми сторонними разработчиками. Имеется возможность

разработки собственного драйвера для самых специфических, и даже, проприетарных версий браузеров.

6. BDD. В дополнение к предыдущему пункту, еще большее распространение фреймворк может получить вследствие возможности разрабатывать автотесты с использованием технологии Behaviour Driven Development, что позволяет даже незнакомым ни с одним языком программирования специалистам в области тестирования ПО, как анализировать текущие написанные тесты, так и разрабатывать новые, используя разработанные базовые шаги. Например, открытие определенной страницы, нажатие, заполнение веб-элементов, проверка текстовых и других данных на страницах, можно делать, не зная традиционных языков программирования.

7. CI/CD. Фреймворк разрабатывается в виде подключаемой библиотеки, что способствует простоте встраивания в свой проект, на уровне прописывания зависимости в настроечных файлах для систем Maven, Ant, Gradle и др. Это упрощает интеграцию в CI/CD среды, такие как Jenkins, Hudson и др.

При этом необходимо отметить и недостатки разрабатываемого решения:

1. Распространенность. На начальном этапе развития, про проект будет известно только узкому кругу внутренних разработчиков и ограниченному кругу пользователей, нашедших open-source репозиторий проекта. Таким образом, поддержка проекта будет осуществляться только силами внутренних разработчиков. Этот недостаток будет нивелирован, в результате дальнейшего распространения инструмента.

2. Язык программирования. Несмотря на то, что выбран самый распространенный на текущий момент язык Java, этим может быть ограничен круг пользователей, предпочитающий другие популярные языки, используемые для разработки автоматизированных тестов для ПО, такие как Python, Ruby и C#. Частично этот недостаток нивелируется возможностью использования средств технологии BDD, и таким образом можно частично избежать

необходимость иметь штат специалистов, знакомых с языком Java. Но полностью исключить такую необходимость не получится.

1.5. Концептуальная модель процесса работы отдела тестирования «Как должно быть»

На основании предложений в параграфах 1.3., 1.4. для увеличения эффективности деятельности отдела тестирования разработана модель «Как должно быть» с использованием фреймворка автоматизации тестирования. Ее схема и декомпозиция процесса представлены на рисунках 1.3 и 1.4.

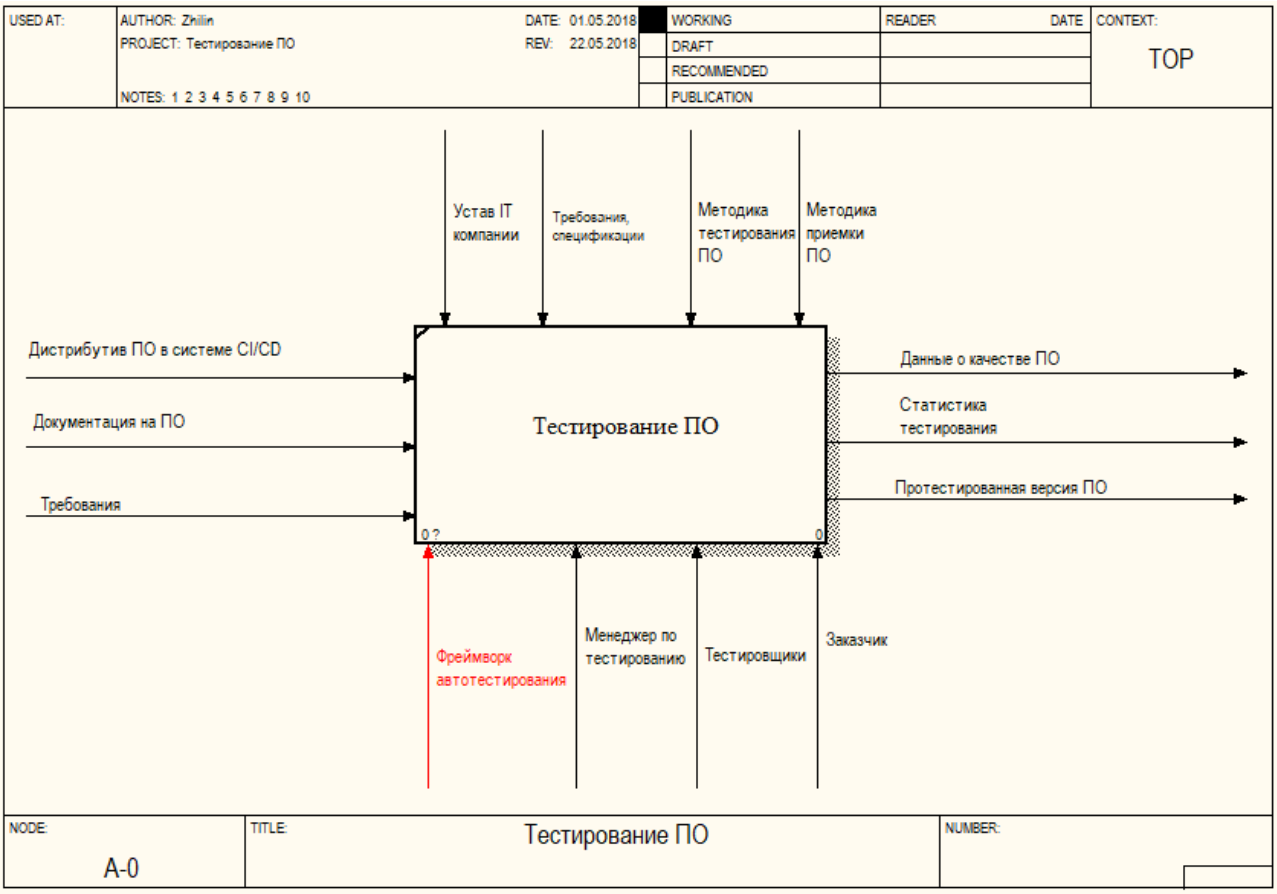

Рисунок 1.3. Схема процесса работы отдела тестирования «Как должно быть»

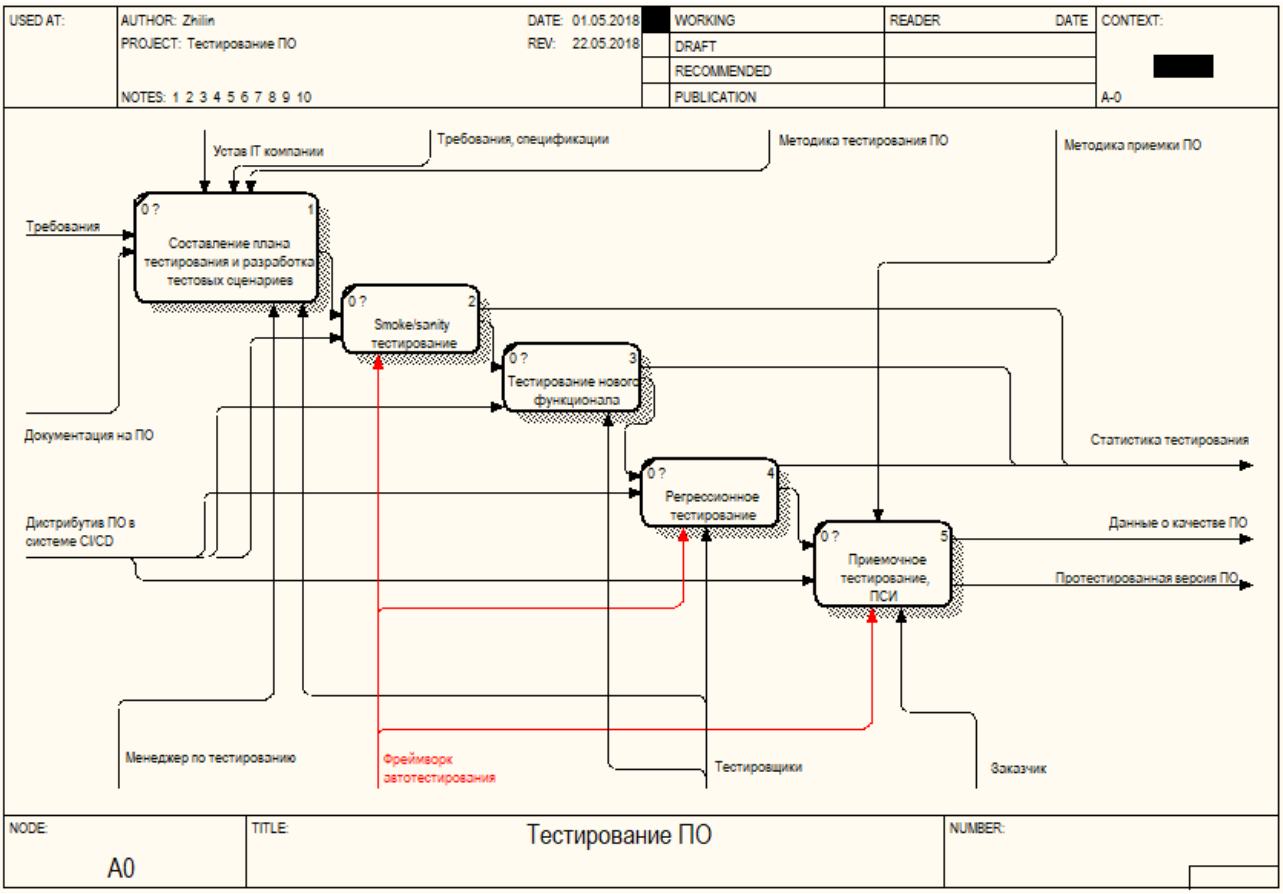

Рисунок 1.4. Декомпозиция схемы процесса работы отдела тестирования «Как должно быть»

Фреймворк будет использован для разработки и прогонов тестов на стадиях smoke/sanity тестирования, регрессионного тестирования, а также по желанию заказчика (внешнего или внутреннего) на стадии приемочного тестирования/приемо-сдаточных испытаний.

Это позволит сократить время, затраченное специалистами по тестированию на прогон тестов, которые производились в ручном режиме, а также обеспечит возможность исключения разработчиков из участия в процессе на этапе smoke/sanity тестирования, так как проверка работоспособности поставленной версии будет проводиться автоматически.

Также фреймворк поможет в сокращении времени на подготовку отчетов о прогонах тестов, так как этот процесс будет также происходить автоматически, и отчеты будут выгружаться в удобной форме на специальный портал, с возможностью связки с инструментами Continuous Integration/ Continuous Deployment.

Таким образом, после внедрения фреймворка автоматизации тестирования, выявленные недостатки будут устранены, а при необходимости наработки по внедрению автоматизированных тестов могут быть переданы заказчику для повышения эффективности прохождения приемочного тестирования на аппаратном окружении заказчика.

Выводы по главе 1

В главе 1 были рассмотрены вопросы необходимости создания программного средства для автоматизации тестирования в виде единого фреймворка, а также проблемы и недостатки, которые можно решить с помощью данного программного средства.

Рассмотрена важность как самого процесса тестирования в рамках деятельности компании разработчика программного обеспечения, так и важность автоматизации тестов.

Путем концептуального моделирования представлены схемы улучшения общего процесса тестирования за счет использования средств автоматизации, а также рассмотрены пути повышения эффективности деятельности отдела/департамента тестирования за счет внедрения фреймворка автоматизированного тестирования.

Была описана постановка задачи по созданию фреймворка с указанием основных целей, моделей решения задачи и порядка работы пользователей с системой.

В заключение первой главы был проведен анализ аналогов, существующих на рынке, и приведена сравнительная характеристика достоинств и недостатков разрабатываемого программного средства по сравнению с действующими продуктами.

Глава 2. Разработка фреймворка автоматизированного тестирования веб-приложений

2.1. Разработка логической модели автоматизации тестирования

2.1.1. Логическая модель фреймворка автоматизированного тестирования и ее описание

Для создания объектной модели фреймворка был выбран язык UML (Unified Modeling Language).

Данный язык позволяет в виде стандартизированных чертежей и схем визуализировать, специфировать, конструировать и документировать программные системы. Язык пригоден для составления моделей систем любого масштаба и направления, его применяют при моделировании информационных системы больших и малых предприятий, Web-приложений, встроенных систем реального времени.

Для построения логической модели данных фреймворка была выбрана методология IDEF1x, которая применяется для создания реляционных моделей данных.

Далее представлен переход от структурной диаграммы «как должно быть», описанной в главе 1 к диаграмме вариантов использования (Use case Diagram), которая также имеет название – диаграмма прецедентов.

В соответствии с руководством пользователя по языку UML «Субъект (actor) – любая сущность, взаимодействующая с системой извне или множество логически связанных ролей, исполняемых при взаимодействии с прецедентами. Стандартным графическим обозначением субъекта на диаграммах является фигура "человечка", под которой записывается конкретное имя субъекта, однако субъектом может быть не только человек, но и техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик» [2].

«Прецеденты (use case) – это описание множества последовательностей действий (включая их варианты), которые выполняются системой для того, чтобы актер получил результат, имеющий для него определенное значение. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие субъектов с системой, это одна из важнейших особенностей разработки прецедентов. Стандартным графическим обозначением прецедента на диаграммах является эллипс, внутри которого содержится краткое название прецедента или имя в форме глагола с пояснительными словами» [2].

Диаграмма прецедентов использования предназначена для описания функционального назначения системы. Диаграмма прецедентов фреймворка автоматизации тестирования приведена на рисунке 2.1.

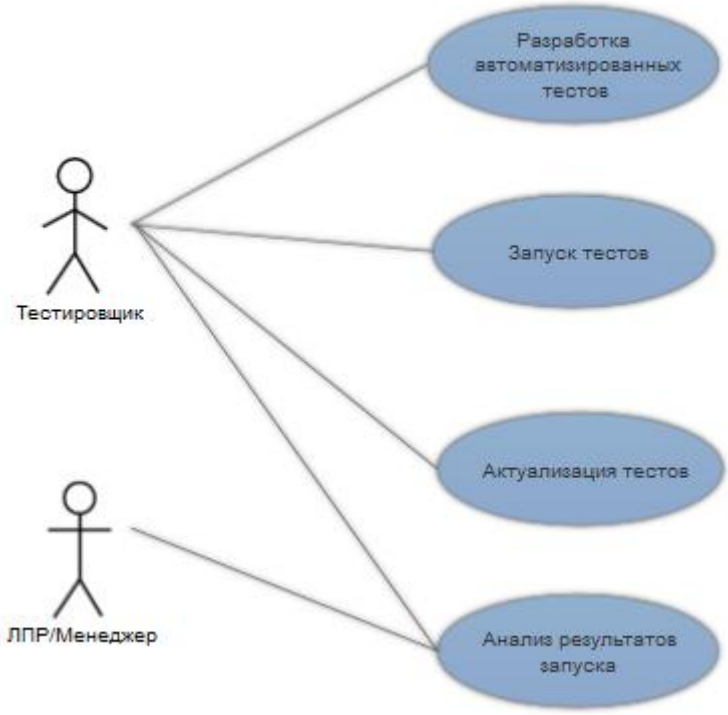

Рисунок 2.1. Диаграмма прецедентов использования фреймворка автоматизации

Далее, разработаем диаграмму классов для фреймворка (Class Diagram).

Класс – это основной составной элемент информационной системы. Данное понятие является одним из основополагающих в объектно-ориентированных языках программирования. Соответственно, между классами

в диаграммах UML и классами в программном коде есть соответствие, поэтому большинство средств построения диаграмм UML позволяют генерировать программный код на выбранном языке программирования на основе диаграммы классов.

Каждый класс обязательно имеет уникальное название, а также атрибуты и методы. Атрибутами называются свойства класса, а методами – действия, которые может выполнять объект выбранного класса, чаще всего методы класса предназначены для изменения его свойств.

Абстрактная диаграмма классов для разрабатываемого фреймворка представлена на рисунке 2.2.

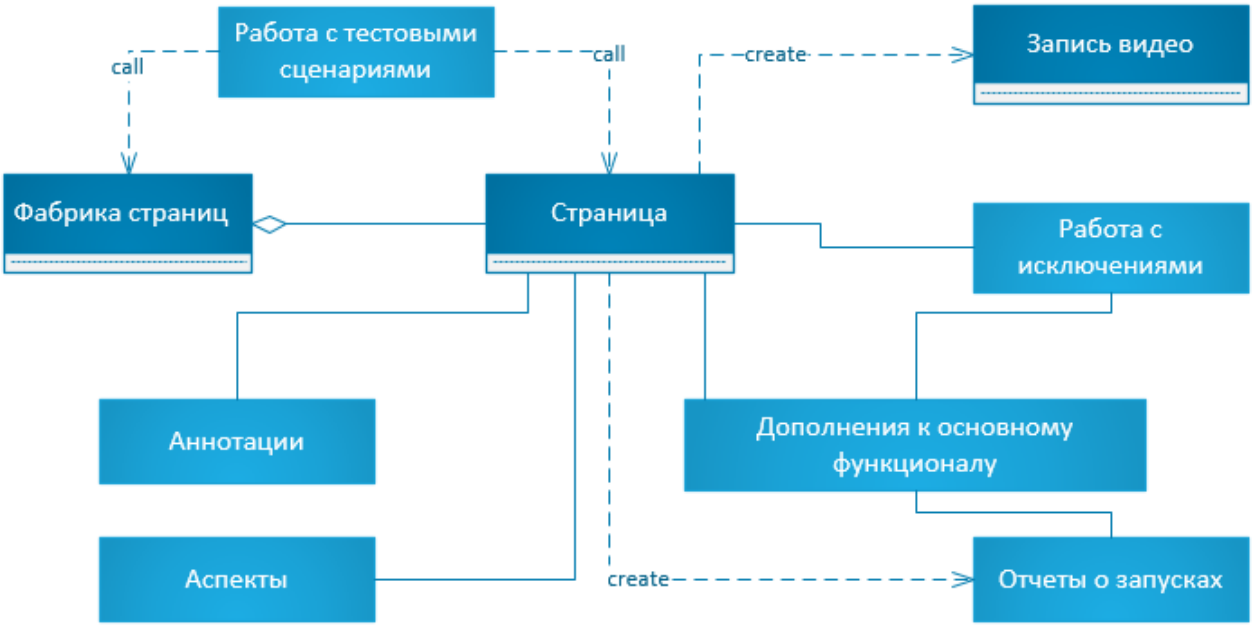

Рисунок 2.2. Абстрактная диаграмма классов фреймворка для автотестирования

Из данной диаграммы видно, какие сущности будут представлены в разрабатываемом фреймворке и их связи. Более подробная логическая реализация представлена в параграфе 2.1.2

2.1.2. Диаграммы классов фреймворка

Рассмотрим более подробно каждый из классов, а также его поля и методы. Для этого классы разделены на основной функционал и служебный

функционал. Основной функционал является ядром деятельности фреймворка, а служебный функционал отвечает за обслуживание основного, либо за предоставление дополнительных функций для удобства разработки тестов.

Основной функционал:

1. Фабрика страниц (PageFactory). См. рисунок 2.3.

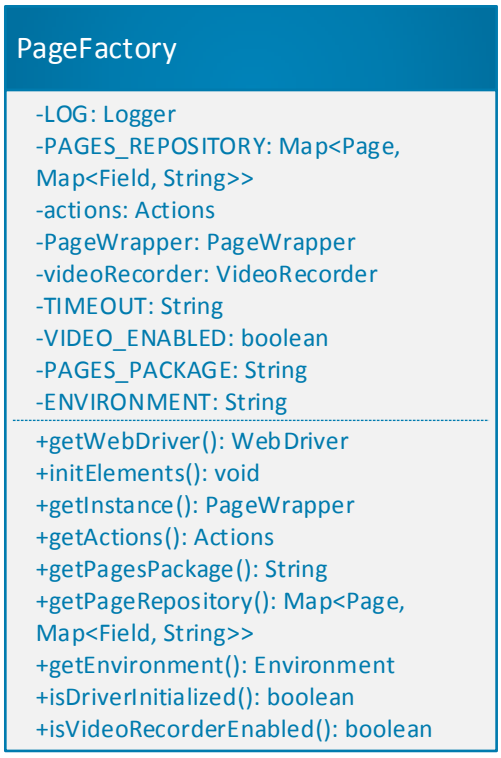

Рисунок 2.3. Фабрика страниц. PageFactory.

Фабрика страниц является основным классом, так как на нем основана вся работа фреймворка. Он реализован в качестве следования концепции PageObject, когда для представления элементов и действий создается класс конкретной страницы (Page). PageFactory является управляющим классом для пакетов страниц, а также отвечает за инициализацию веб-драйвера, который будет проводить тесты в конкретном браузере и отвечает за запись видео процесса тестирования.

Основными полями являются:

- LOG – стандартная переменная класса Logger, относящегося к библиотеке логирования slf4J. Данное поле используется практически во всех остальных классах, поэтому далее описываться не будет.

- PAGES_REPOSITORY – хранилище (Map) классов страниц.

- actions – ссылка на класс действий с веб-страницей из библиотеки Selenium.

- PageWrapper – класс, отвечающий за получение страниц и их названий классов.

- videoRecorder – ссылка на объект класса VideoRecorder, отвечающий за запись видео.

Основные методы класса PageFactory:

- getWebDriver() – получение ссылки на веб-драйвер;
- getInstance() – получение ссылки на PageWrapper для работы с конкретными страницами;
- getActions() – получение ссылки на объект с возможными действиями со страницами;
- initElements() – инициализация элементов на странице.

2. Страница (Page). См. рисунок 2.4.

Рисунок 2.4. Страница. Page.

Страница (Page) является родительским классом для всех будущих разрабатываемых конкретных страниц. В данном классе содержатся основные действия, которые можно совершать с веб-страницами.

Основным полем является title – наименование страницы, используемое для ее идентификации.

Основные методы класса Page:

- fillField(WebElement, String) – заполнение текстового поля поданным значением. Когда в модели указан тип WebElement – это значит, что будет использован класс WebElement из библиотеки Selenium, который используется для работы с конкретными веб-элементами.
- clickWebElement(WebElement) – имитация нажатия на элемент.
- pressKey(Keys) – имитация нажатия конкретных клавиш на клавиатуре.
- setCheckBoxState(WebElement, boolean) – установка, либо снятие чекбокса.
- select(WebElement, String, MatchStrategy) – выбор элемента из какого-либо списка на странице. Аргумент MatchStrategy отвечает за стратегию поиска текста элемента (например, точное соответствие, содержит, начинается с, и т.п.).
- acceptAlert(String) – подтверждение всплывающего окна нажатием кнопки «Ок».
- dismissAlert(String) – отмена всплывающего окна нажатием кнопки «Отмена».
- и другие методы, такие как проверка содержимого элементов, получение значений параметров элементов, выполнение методов, поиск элементов на странице.

3. Обертка для страницы (PageWrapper). См. рисунок 2.5.

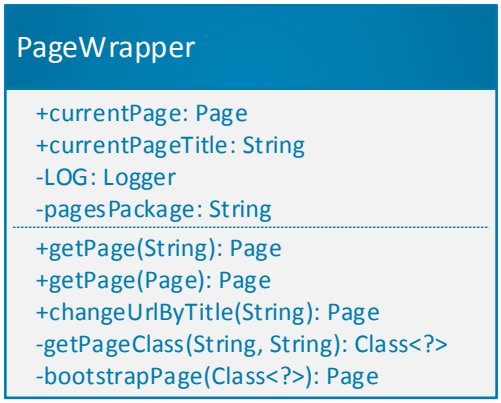

Рисунок 2.5. Обертка для страницы. PageWrapper.

Данный класс служит как обертка для работы с конкретными страницами. Используется в тех случаях, когда неизвестно какая точно страница открыта и позволяет вызывать методы, общие для всех страниц, для текущей страницы, открытой в браузере.

Основными полями являются:

- currentPage – ссылка на объект текущей страницы, открытой в браузере.
- currentPageTitle – заголовок текущей страницы.

Основные методы класса PageWrapper:

- getPage() – получение текущей страницы по ссылке, либо по заголовку.
- changeUrlByTitle() – смена текущего URL страницы по заголовку.

4. Работа с тестовыми сценариями - базовые шаги автотестов (GenericStepDefs). См. Рисунок 2.6.

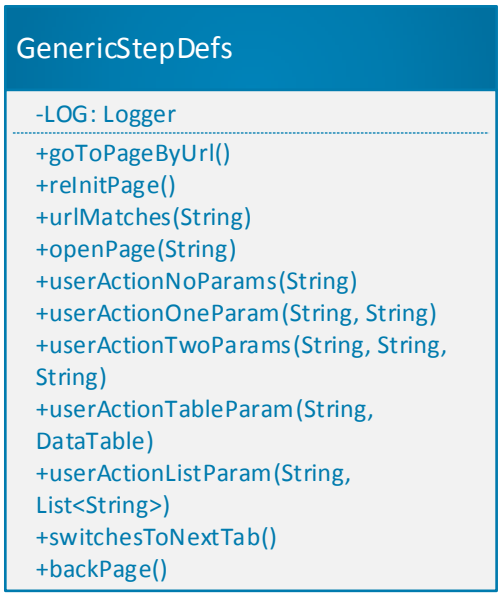

Рисунок 2.6. Базовые шаги автотестов. GenericStepDefs.

Данный класс служит основой для программирования шагов автотестов с использованием библиотеки Cucumber в стиле BDD. Соответственно, все методы данного класса будут аннотированы аннотацией @And библиотеки Cucumber для использования человекочитаемого языка при описании сценариев тестирования.

Основные методы класса GenericStepDefs:

- gotoPageByUrl() – переход на определенную страницу по URL.
- reInitPage() – переинициализация страницы.
- openPage() – открытие страницы по заголовку.
- userAction() – вызов заранее прописанных на странице действий с разными наборами тестовых данных: одно значение, 2 значения, таблица значений, либо список значений.
- switchesToNextTab() – переключение на следующую вкладку в браузере.

5. Запись видео выполнения тестов (VideoRecorder). См. рисунок 2.7.

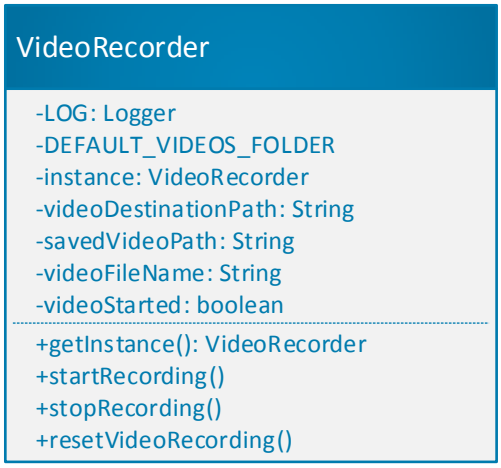

Рисунок 2.7. Видео рекордер. VideoRecorder.

Данный класс служит для записи видео выполнения тестовых сценариев в браузере.

Основными полями являются:

- DEFAULT_VIDEOS_FOLDER – директория по умолчанию для записей видео.

- instance – ссылка на текущий объект рекордера.

- savedVideoPath – установленная директория для сохранения видео-файлов.

- videoFileName – имя файла с текущим сохраняемым видео.

Основные методы класса VideoRecorder:

- getInstance() – получение ссылки на текущий объект рекордера.
- startRecording() – старт процесса записи видео.
- stopRecording() – остановка процесса записи видео.
- resetVideoRecording() – сброс текущих значений для записи видео.

Служебный функционал:

1. Драйверы для работы с Selenium WebDriver и AppiumDriver (для мобильных веб-приложений). См. рисунок 2.8.

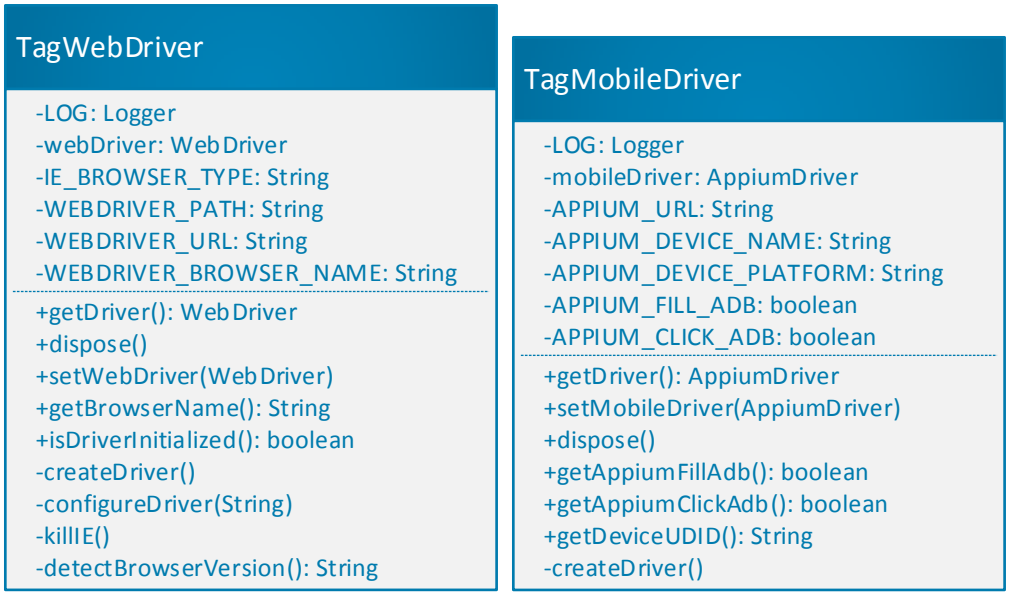

Рисунок 2.8. Драйверы для работы с браузерами.

Данные классы предназначены для работы с основным инструментарием Selenium WebDriver для десктопных веб-приложений и AppiumDriver для мобильных веб-приложений.

Основными полями являются идентификационные данные по связи драйвера с реальным браузером.

Основные методы: создание драйвера, конфигурация драйвера, получение данных о среде выполнения, закрытие браузера.

2. Расширения к веб-драйверу (DriverExtension). См. рисунок 2.9.

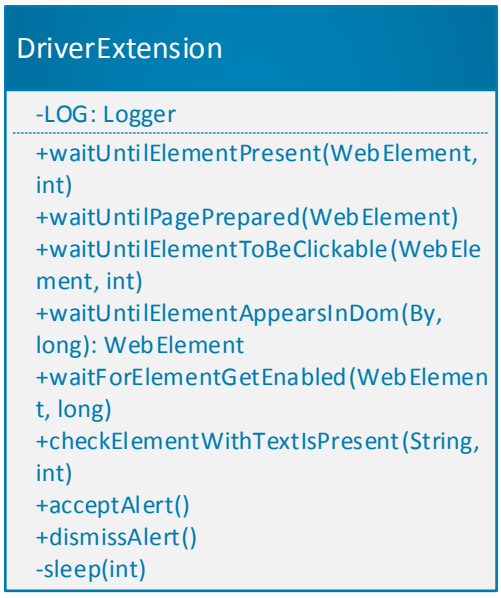

Рисунок 2.9. Расширенные функции для работы с драйверами.

Данный класс служит для расширения возможностей по работе с веб-драйверами.

Основные методы класса DriverExtension:

- waitUntilElementPresent(WebElement, int) – ожидание появления элемента на странице с определенным таймаутом.
- waitUntilPagePrepared(WebElement) – ожидание загрузки страницы, опираясь на определенный базовый элемент.
- waitUntilElementToBeClickable(WebElement, int) – ожидание «кликабельности» определенного элемента.
- checkElementWithTextIsPresent(String, int) – проверка наличия на странице веб-элемента с определенным текстовым значением.
- sleep(int) – явное ожидание определенное количество секунд без каких-либо действий.

3. Аспекты (Aspect). См. рисунок 2.10.

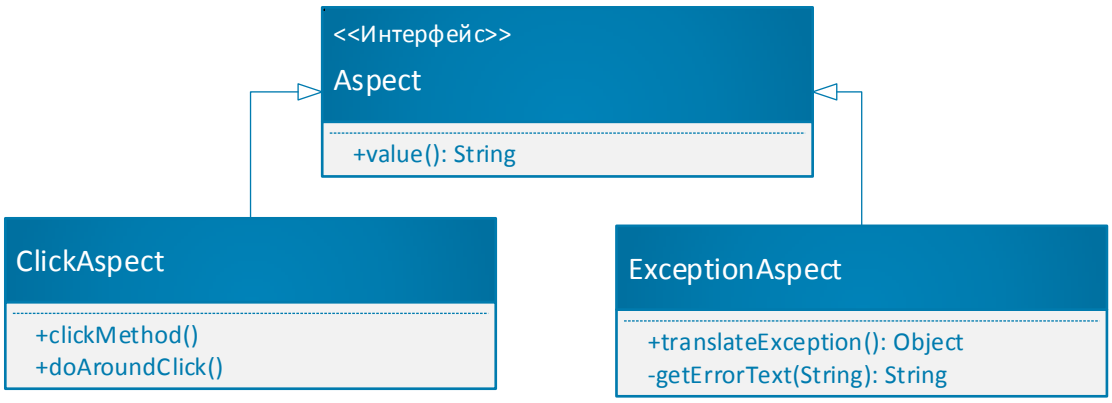

Рисунок 2.10. Аспекты для определенных действий.

Данные классы являются наследниками интерфейса Aspect из библиотеки aspectj и служат для совершения действий вместе с методами, не определенными явно, а выявляемыми по определенным шаблонам. В данном фреймворке разработаны аспекты для вызовов при отработке методов, связанных с нажатием всех элементов на веб-страницах - ClickAspect, а также при выбрасывании программой любых исключений - ExceptionAspect.

4. Аннотации (Annotation). См. рисунок 2.11.

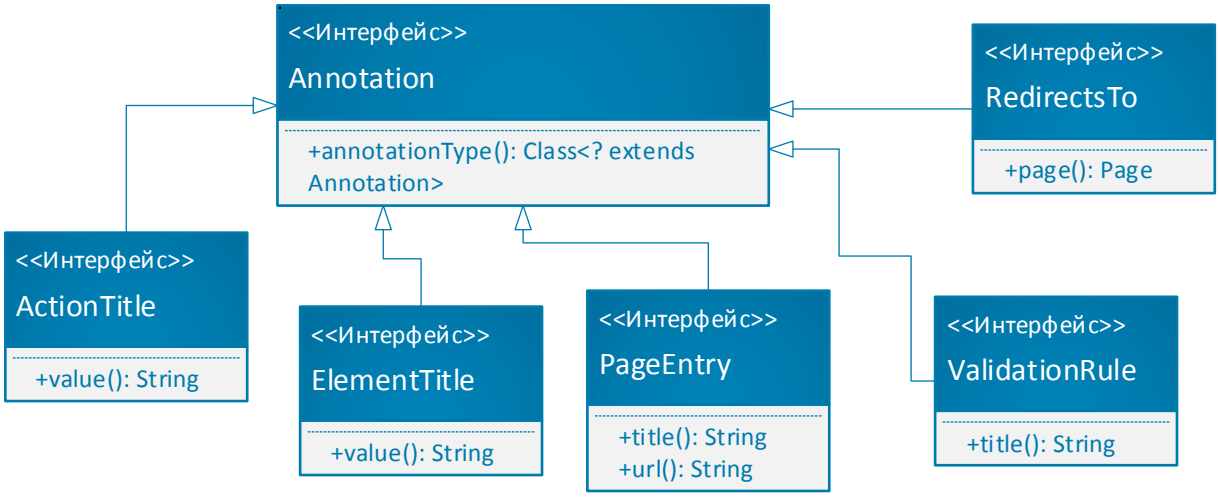

Рисунок 2.11. Интерфейсы аннотаций для элементов и методов.

Данные интерфейсы используются для аннотирования методов и полей с целью идентификации для получения данных, либо для совершения определенных действий.

ActionTitle – служит для обозначения методов на конкретных страницах в человекочитаемой форме, в том числе на русском языке.

ElementTitle – служит для идентификации элементов также в словесной форме.

PageEntry – служит для идентификации веб-страниц.

ValidationRule – служит для обозначения того, какие методы валидации значений будут применяться к определенным полям.

RedirectsTo – определяет на какую страницу будет осуществляться переадресация.

5. Исключения (Exception). См. рисунок 2.12.

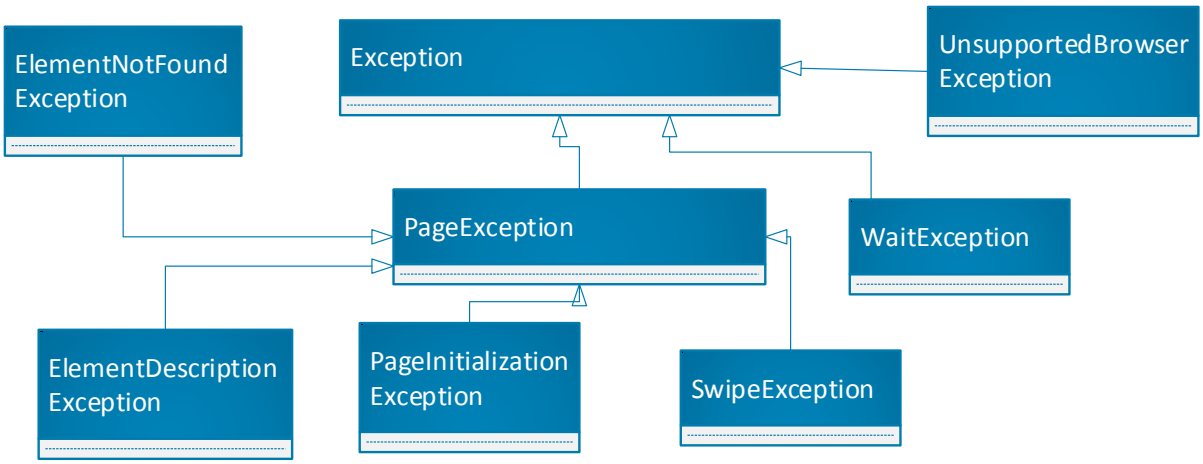

Рисунок 2.12. Классы исключений.

Для фреймворка разработаны специфические классы исключений для увеличения понятности для конечного пользователя.

Класс PageException служит для выбрасывания исключений при работе с веб-страницами, и включает в себя дочерние, более специализированные подклассы:

ElementNotFoundException – выбрасывается при не нахождении элемента на веб-странице по определенному локатору.

ElementDescriptionException – выбрасывается при несовпадении данных, ожидаемых от определенного элемента.

PageInitializationException – выбрасывается при невозможности проинициализировать веб-страницу в браузере.

SwipeException – выбрасывается при ошибках работы с мобильными браузерами во время действия «swipe».

Кроме этого, разработаны исключения WaitException и UnsupportedBrowserException, выбрасываемые при ошибках ожидания и ошибках неподдерживаемого браузера, соответственно.

Общая диаграмма классов со связями между ними представлена в параграфе 2.2.3.

2.1.3. Характеристика входной оперативной информации

Для фреймворка автоматизированного тестирования входной информацией являются сценарии тестирования (тест-кейсы) и тестовые данные, необходимые для их выполнения.

Сценарии тестирования представляют собой сущности, состоящие из следующих составных частей:

- название – название тест-кейса, идентифицирующее его назначение;

- описание – здесь может быть представлено развернутое описание и комментарии по выполнению тестового сценария;

- ссылка на сценарий в системе ведения тест-кейсов – служит для связи кейсов, результатов их выполнения с системой заведения тест-кейсов в текстовой форме;

- предусловие – необходимые условия выполнения тестового сценария, которые необходимо предусмотреть для его успешного выполнения;

- шаги – описание шагов процесса прохождения сценария, включая действие и описание ожидаемого результата данного действия;

- постусловия – описываются действия, которые необходимо, либо желательно совершить в конце после окончания исполнения сценария, для приведения системы в актуальное состояние.

Сценарии тестирования в разрабатываемом фреймворке планируется принимать в форме, соответствующей методологии BDD (Behaviour Driven Development). В соответствии с данной методологией, сценарий тестирования описывается в следующем формате:

Допустим (Given) некоторые начальные данные,

Если (When) возникает или запускается какое-то событие,

То (Then) получаем определенный результат.

Данная схема позволяет довольно легко переходить от сценария, описанного в текстовом виде к сценарию, описанному для автоматического выполнения фреймворком. Для этого следует изучить базовые текстовые конструкции, которые затем используются как модульные наборы для составления полноценных тестовых сценариев.

Приведем пример тестового сценария для формы авторизации:

Допустим Открыта страница авторизации

Если Ввести логин «инспектор»

И Если ввести пароль «12345»

И Если нажать кнопку «Вход»

То Открыта страница «Главное меню».

Тестовые данные представляют собой информацию, которую требуется вводить на различных формах графического интерфейса, либо использовать для посылки в веб-сервисы и процедуры. Также тестовыми данными могут служить настроечные данные, такие как, ссылки на шаблоны для формирования файлов, ссылки на файлы с примерами для сравнения верстки, печатных форм и т.п.

Тестовые данные можно задавать в большом количестве форматов, например:

- в виде файлов Microsoft Excel;
- в виде файлов с разделителями CSV (comma separated values);
- в виде файлов в формате XML;
- в виде файлов в формате json;
- в виде хранения информации в реляционной, либо noSQL базе данных.

Итоговый формат тестовых данных выбирается исходя из удобства для конечных пользователей, а также может зависеть от уровня их технической подготовки. В рамках фреймворка не реализована поддержка конкретных форматов, однако путем использования библиотек для парсинга вышеуказанных файлов, для любого разработчика не составит труда

представить тестовые данные во внутренней структуре фреймворка для хранения в древовидной форме – Stash (структура описана более подробно в параграфе 2.3.1).

2.1.4. Характеристика результатной информации

Результатная информация, поступающая по результатам работы разрабатываемого фреймворка, представляет собой сведения о прогонах тестов, представленные в разных формах для разных категорий пользователей.

Результаты будут представляться в следующих формах:

1. Логи. Логирование представляет собой низкоуровневые записи о порядке выполнения инструкций кода в процессе выполнения определенных методов. Логирование осуществляется в консоль, либо в виде записи в текстовые файлы. Логирование можно разделять на уровни, отличающиеся более высокой или низкой подробности логируемых действий для различных целей – начиная с целей отладки тестов, и заканчивая целью просмотра краткой информации о том, какие тесты и методы были запущены. Пользователями данной категории результирующей информации будут являться, в основном, разработчики автотестов и тестировщики, обладающие продвинутым уровнем знаний в языке программирования, на котором реализован фреймворк. Логирование во фреймворке осуществляется практически каждым классом, используя объект Logger библиотеки slf4j.

2. Визуальный отчет о результатах конкретных прогонов тестов. Визуальный отчет служит для выведения информации для более широкого круга пользователей, так как не предполагает наличия у пользователя глубоких знаний о внутренней структуре фреймворка. Данный отчет отображает, обычно в формате интерактивной веб-страницы, результаты прогонов конкретных тестов, тестовых наборов, а также общую статистику. Общая статистика обычно включает: время исполнения набора и каждого конкретного теста, количество успешно и неуспешно пройденных тестов, и описание успешно пройденных и неуспешно пройденных шагов для каждого конкретного теста.

Пользователями данной категории результирующей информации будут являться все участники процесса разработки: разработчики автотестов, тестировщики, лица принимающие решения/менеджеры, разработчики тестируемого ПО. Фреймворком предусмотрено подключение отчетов Allure 2.0 с использованием класса TagPFAllureReporter, наследуемый от класса библиотеки Allure – AllureCucumber2Jvm. См. рисунок 2.13.

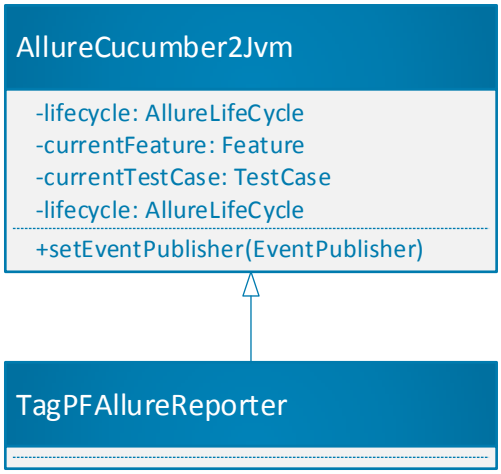

Рисунок 2.13. Класс для работы с выгрузкой результатов в отчет

3. Отправка результатов в ПО для ведения тест-кейсов. Данный вид результирующей информации представляет собой отправку результатов выполнения отдельных тестов, либо тестовых наборов целиком, в специализированное ПО для учета результатов выполнения тестов. Таким ПО может являться, например, IBM Jazz, HP ALM, Atlassian JIRA, Zephyr, и другие программные комплексы, у которых имеется интерфейс/API для приемки результатов выполнения тестов и связь их с заведенными в систему тестами по определенным идентификаторам. Реализация связи с данными инструментами выходит за рамки данной работы, но является технически реализуемой.

2.2. Физическое моделирование фреймворка автоматизированного тестирования

2.2.1. Выбор архитектуры фреймворка

В качестве основы для разработки фреймворка была выбрана методология BDD (Behaviour Driven Development), при этом используется паттерн проектирования PageFactory.

Паттерн проектирования PageFactory заключается в том, чтобы писать автотесты на человекочитаемом языке, тем самым понижая входной порог для разработчиков тестов и повышая их читаемость неподготовленными пользователями. Разрабатываемый фреймворк использует BDD фреймворк Cucumber-JVM, но, в отличие от чистого использования, в котором довольно большую часть архитектуры занимают шаги (Step Definitions), здесь акцент сделан на то, чтобы избавиться от необходимости писать их самому или сократить количество самописных шагов, сосредоточившись на описании кода страниц с использованием паттерна PageObject.

PageObject является самым распространенным паттерном для автоматизированного тестирования графических интерфейсов веб-приложений. В его основе лежит идея, что для каждого экрана (страница, окно) в приложении с автотестами создается своя страница (Page), где описываются все элементы страницы (Controls, Elements). Основным преимуществом данного паттерна является простота поддержки автотестов, так как в случае изменения идентификатора объекта элемента страницы, изменения нужно внести в одном месте, на странице, где он описан.

Использование же BDD позволит писать сценарии автоматизированного тестирования на человекочитаемом языке, что позволяет понизить входной порог для разработчиков автотестов, и позволяет людям не знакомым с языками программирования и автоматизацией тестирования понять и воспроизвести шаги автотеста вручную.

Фреймворк будет разработан как кроссплатформенный, то есть можно будет запускать тесты под всеми операционными системами и на всех

браузерах, которые поддерживаются драйверами Selenium WebDriver, который выбран основной системой управления браузерами.

Также фреймворк позволяет управлять мобильными браузерами и некоторыми мобильными приложениями, используя инструмент Appium для операционной системы Android.

Языком программирования, на котором будет разработан фреймворк, выбран язык Java, как самый распространенный на сегодняшний день по ряду индексов [14].

В качестве аппаратной платформы для запуска тестов с использованием фреймворка достаточно любого компьютера/сервера/виртуальной машины с минимальной конфигурацией, способной загружать современные версии браузеров и виртуальную машину Java.

Минимальная необходимая конфигурация:

- Процессор – Intel Pentium 4 или аналоги с наличием инструкций SSE2.
- Оперативная память – от 2 Гб (рекомендуется 4 Гб).
- Наличие места на диске – минимум 1 Гб (рекомендуется не менее 10 Гб при необходимости длительного хранения отчетов, логов, приложений, скриншотов и видеозаписей).
- Операционная система – Windows от Windows Server 2012 и Windows Vista, Linux, Mac OS X 10.8.3+.

2.2.2. Функциональная схема проекта

Функциональная структура фреймворка включает основные функции, такие как: создание объектов страниц (Page), совершение действий над ними в подключенных браузерах, проверки (Assertions) и др.

Также функциональная структура проекта включает служебные внутренние функции, такие как: управление веб-драйвером, формирование отчетных данных, запись видео, снятие скриншотов.

Более подробно, состав основного функционала:

1. Действия для конкретной страницы:

- ожидание открытия страницы;

- заполнение полей данными;

- нажатие на кнопки и нажимаемые веб-элементы;

- получение текстовых и других параметров и свойств веб-элементов;

- эмуляция нажатия кнопок клавиатуры и мыши;

- подтверждение, отмена действий в модальных окнах;

- проверка наличия веб-элементов на странице;

- проверка видимости/невидимости веб-элементов на странице;

- получение заголовка страницы;

- отправка файлов в контекстных окнах;

- выбор элементов в списках, чекбоксах и радиобаттонах.

2. Реализация основных действий пользователя в рамках подхода BDD/Cucumber (Step Definitions):

- открытие страницы;

- выполнение конкретных действий, описанных в пункте 1 данного раздела;

- выполнение действий в определенных блоках элементов;

- выполнение параметризованных действий, то есть действий с использованием некоторых заранее заданных или сгенерированных тестовых данных;

- переключение между окнами/страницами;

- переход по новой ссылке URL;

- фокусировка и перемотка страницы до конкретного элемента;

- закрытие страницы и браузера.

Описанные в данном пункте шаги реализованы как Step Definitions для фреймворка Cucumber JVM, что позволяет использовать большинство необходимого функционала при написании тестов, непосредственно, без дополнительного программирования.

Более подробно, состав служебных функций:

1. Действия для работы с веб-драйвером Selenium в рамках подхода PageFactory:

- инициализация веб-драйвера и связь с конкретным браузером;

- получение пакета классов страниц, которые будут использованы в текущих тестах;

- получение класса конкретной страницы и инициализация элементов на ней;

- получение информации об окружении и текущих настройках;

- организация хранения тестовых данных;

- получение информации о состоянии браузера;

- настройка параметров браузера;

- включение/выключение записи видео процесса в браузере;

- отключение веб-драйвера и закрытие окна или вкладки браузера;

- ожидания для разных действий.

2. Функции для работы с исключениями:

- исключения, связанные с инициализацией браузера;

- исключения, связанные с неподходящими версиями браузера для конкретных тестов;

- исключения, связанные с поиском и инициализацией конкретных элементов;

- ошибки автотестов.

3. Работа с аспектами, то есть функции, которые будут исполняться совместно с определенными функциями, обозначенными специальным шаблоном:

- аспекты для нажатия на элементы;

- аспекты для работы с исключениями.

4. Функции для работы с аннотациями, представляющими собой специальные пометки для методов и конкретных переменных объектов:

- аннотации для поиска и идентификации веб-элементов;

- аннотации для обозначения отдельных функций для работы со страницей;

- аннотации, описывающие правила валидации для конкретных элементов.

Таким образом, функционально проект полностью соответствует основным потребностям пользователей, которым необходимо автоматизировать функционал тестирования веб-приложения/сайта. При этом, в случае отсутствия конкретных специфических функций, их легко можно реализовать, либо используя механизм ActionTitle'ов на конкретной странице, либо путем написания отдельного шага для выполнения конкретного действия, либо набора действий.

2.2.3. Описание программных модулей фреймворка автоматизированного тестирования

В данном разделе представлена схема классов, их взаимодействия и взаимозависимостей. Полная схема, частично свернутая по пакетам, представлена рисунке 2.14.

Из данной схемы видно, что класс PageFactory является главным управляющим классом, который управляет веб-драйверами, страницами и должен получать определения шагов для тестов для их дальнейшего выполнения из пакета Package stepdefs.

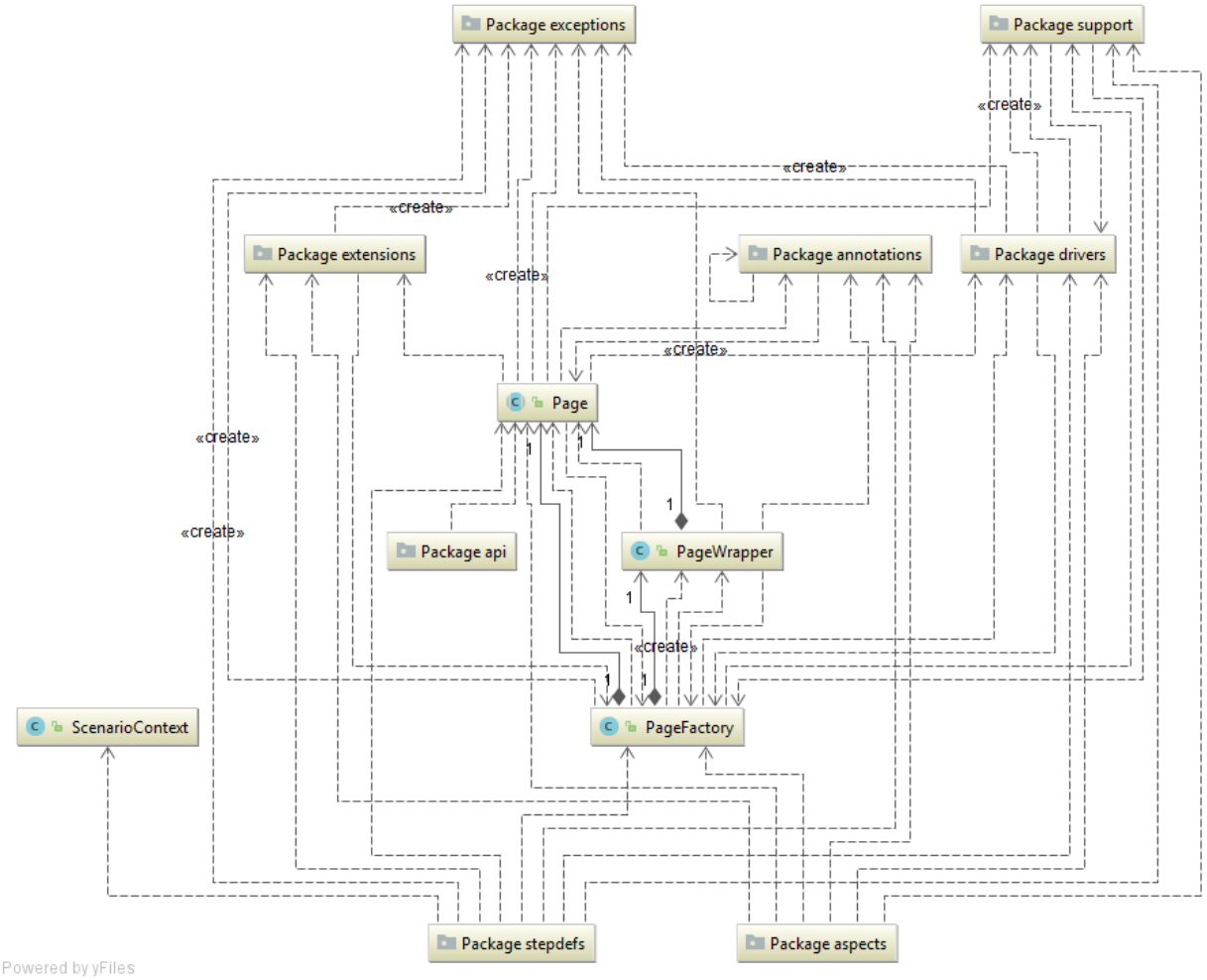

Рисунок 2.14. Схема классов и пакетов фреймворка автотестирования

Класс Page является базовым классом выше описанного паттерна PageObject и управляет всей работой с веб-страницами. От данного класса следует наследовать конкретные страницы тестируемого веб-приложения, наполняя их элементами и необходимыми действиями.

Практически из всех классов проекта могут выбрасываться исключения при выполнении тестов. За их обработку отвечает пакет классов Package exceptions.

Аннотации в пакете Package annotations влияют на работу некоторых методов и работу с конкретными переменными, что позволяет реализовывать удобную идентификацию, в том числе на местных языках, без привязки к ограничениям языка программирования Java.

Класс PageWrapper является оберткой, позволяющей оборачивать классы страниц в специальные блоки для их более удобной идентификации.

Практически из всех классов данные могут попадать к классам пакета Package support. В данном пакете реализованы классы для сохранения скриншотов конкретных страниц и действий на них, а также реализован класс для подключения системы генерации отчетов Allure.

2.3. Технологическое обеспечение задачи

2.3.1. Организация технологии сбора, передачи, обработки и выдачи информации

В рамках фреймворка сбор информации представляет собой получение готовых тестовых сценариев, для которых предполагается исполнение. Подача такой информации может быть реализована, например, на основе библиотек для unit-тестирования, таких как Junit, Testng. Связь классов для запуска с тестовыми сценариями может осуществляться путем определения ярлыков конкретных сценариев, либо ярлыков целых блоков, содержащих сценарии, объединенные по какому-либо признаку. Также в аннотации к классу запуска можно прописать связи с конкретными пакетами шагов, как реализованных во фреймворке, так и реализованных самостоятельно, в дополнение к основным шагам.

Пример связи класса запуска со сценарием и с набором базовых шагов:

Класс запуска (см. рисунок 2.15):

```
1   package ru.andzhil;
2
3   import ...
6
7   @RunWith(FrameworkCucumber.class)
8   @CucumberOptions(monochrome = true, format = {"pretty"},
9           glue = {"ru.andzhil.autoframework.stepdefs"},
10          features = {"src/test/resources/features/"},
11          tags = {"@ymtest"})
12  public class CucumberTest {}
```

Рисунок 2.15. Класс запуска CucumberTest

Сценарий (см. рисунок 2.16):

YandexMarket.feature

```
# language:ru
Функционал: Тестирование поиска Yandex Market
  @ymtest            #ярлык для связки
  Сценарий: Просмотр результатов поиска
    Когда пользователь переходит на страницу "http://market.yandex.ru" по ссылке
    Тогда открывается страница "Яндекс Маркет"
    И пользователь в блоке "Меню поиска" (выполняет поиск) с параметром "Пылесосы"
    Тогда открывается страница "Результаты поиска товаров"
    И пользователь (проверяет присутствие продукта) "KARCHER WD"
```

Рисунок 2.16. Сценарий запуска теста

При этом сценарий должен лежать в папке «src/test/resources/features», прописанной параметром features в аннотации опций класса запуска.

Тестовые данные могут подаваться как в рамках прописывания в тестовых сценариях, так и другими путями, например с помощью xls, xml, csv, json файлов, либо путем хранения в базе данных. Механизм подачи и хранения тестовых данных должен быть реализован конкретными пользователями фреймворка, так как создать в данном случае универсальный инструмент, подходящий всем, фактически труднореализуемо.

Для хранения тестовых данных и просто временных данных, возникающих во время выполнения теста, может использоваться структура, которая имеет название Stash. Stash представляет собой структуру данных типа HashMap, в которой ключами будут являться названия блоков данных, а значениями – содержимое этих блоков. Содержимым блоков могут являться как конкретные пары ключ-значение, так и вложенные блоки данных, что может обеспечить произвольное количество уровней вложенности и позволяет получить удобную древовидную структуру временного хранения данных в памяти среды выполнения фреймворка и тестов.

Для формирования отчетов предполагается подключение фреймворка генерации отчетов Allure. Данный фреймворк обладает широкими возможностями просмотра результатов выполнения тестов, с возможностью приложения видеозаписей, файлов, скриншотов, логов и т.п. Также фреймворк обладает функцией генерации удобного веб-интерфейса для просмотра отчетов,

а также функциями интеграции с инструментами Continuous Integration, такими как Jenkins/Hudson.

Путем использования среды Jenkins удобно производить настройку запуска тестов, создавая объекты типа job для запуска конкретных наборов тестов с использованием фреймворка. В данной среде удобно производить параметризованные запуски для тестов, например, для разных браузеров, либо с разными настройками, не требуя при этом каждый раз заново перенастраивать среду запуска.

Инструмент Jenkins позволяет также использовать стек компьютеров, либо виртуальных машин, доступных и пригодных для запуска тестов, то есть пользователю не придется заботиться о наличии свободной машины для запусков. Машина будет выделена автоматически в зависимости от использования и нагрузки. При этом можно настраивать ярлыки для машин, которые настроены под конкретные группы тестов.

2.4. Контрольный пример реализации проекта и его описание

В данном параграфе представлен пример работы настроенного тестового сценария, который проверяет страницу поиска продукта в системе «Яндекс.Маркет». На следующих рисунках представлен последовательный процесс создания тестового сценария, его запуска, а также формирования результата с помощью системы генерации отчетов Allure 2.0.

В начале, создается новый Java проект и в нем настраивается связь с библиотекой фреймворка либо подключением как локального модуля, либо с помощью прописывания зависимости в системах Maven/Gradle.

Далее, необходимо создать файл application.properties в директории «test/resources/config». Данный файл содержит настройки фреймворка, такие как: ссылка на пакет с классами страниц, настройки таймаутов, настройки записи видео, настройки веб-драйвера: запускаемый браузер, начальная открываемая страница и другие. Пример файла на рисунке 2.17.

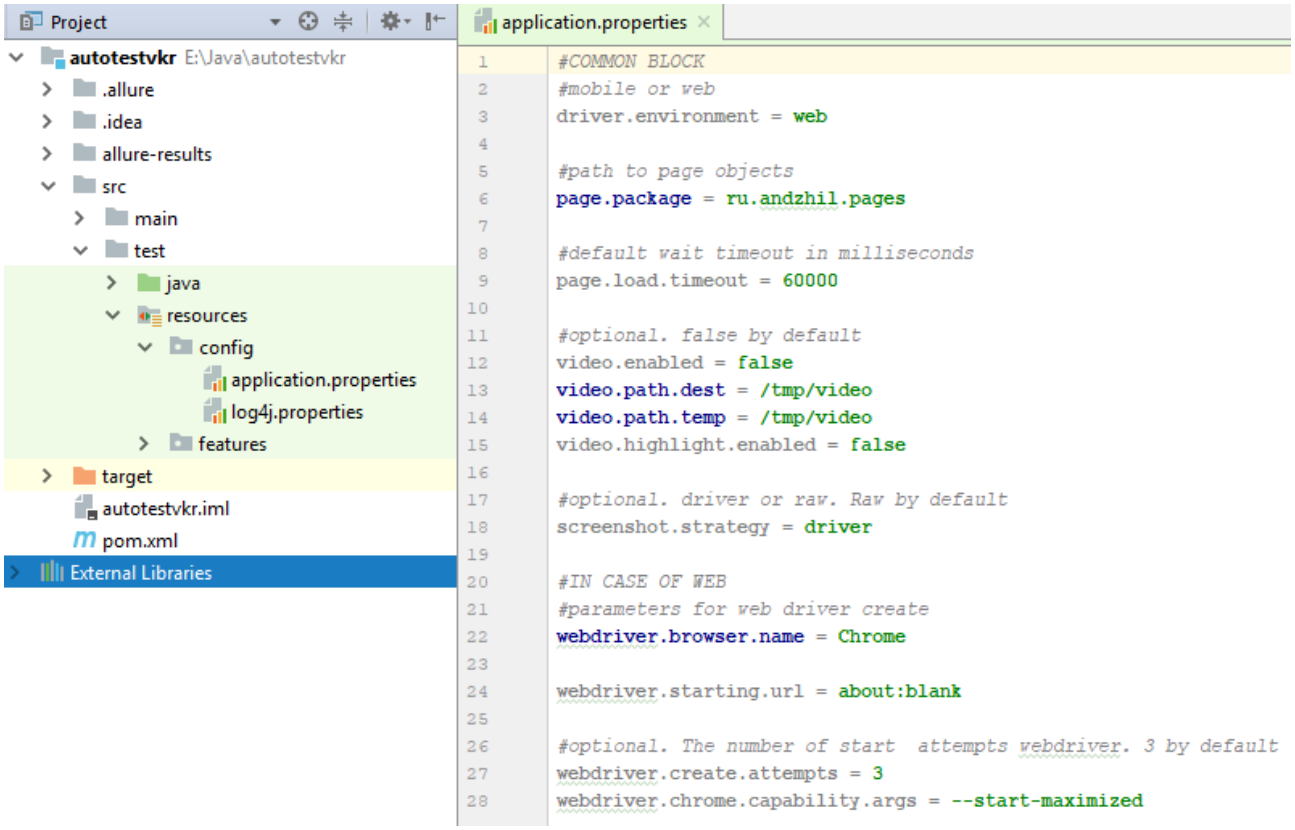

Рисунок 2.17. Пример файла настроек фреймворка application.properties

После этого создается в той же директории файл с настройками системы логирования log4j.

Далее создаются необходимые классы конкретных страниц в отдельном пакете, например, pages. В случае данного примера, это будут 2 страницы: YmMainPage – основная страница сервиса «Яндекс.Маркет» и YmSearchResultsPage – страница с результатами поиска продуктов на «Яндекс.Маркете». В данных классах определяются необходимые для теста элементы.

На странице YmMainPage – это элемент «Меню поиска», вынесенный в отдельный блок HeaderBlock с элементами:

- строка поиска;

- кнопка «Найти».

Для блока Header Block также определено действие с помощью аннотации @ActionTitle – выполнение поиска (см. рисунки 2.18, 2.19).

YmMainPage.java

```java
package ru.andzhil.pages.YandexMarket;

import ...

@PageEntry(title = "Яндекс Маркет")
public class YmMainPage extends Page {

    @ElementTitle("Меню поиска")
    public HeaderBlock headerBlock;

    public YmMainPage() {
        PageFactory.initElements(
                new HtmlElementDecorator(
                        new HtmlElementLocatorFactory(
                                PageFactory.getDriver())), page: this);
    }
}
```

Рисунок 2.18. Класс основной страницы сервиса «Яндекс.Маркет»

YmMainPage.java | HeaderBlock.java

```java
package ru.andzhil.blocks.YandexMarket;

import ...

@Name("Меню поиска")
@FindBy(xpath = "//div[@class='header2__main']")
public class HeaderBlock extends HtmlElement{

    @Name("Строка поиска")
    @FindBy(xpath = ".//input[@id='header-search']")
    public TextInput searchString;

    @Name("Найти")
    @FindBy(xpath = ".//span[text()='Найти']")
    public TextInput searchButton;

    @ActionTitle("выполняет поиск")
    public void startSearch(String text){
        searchString.sendKeys(text);
        searchString.sendKeys(Keys.ENTER);
    }
}
```

Рисунок 2.19. Класс блока меню поиска

На странице YmSearchResultsPage определены элементы:

- блок поиска – ссылка на тот же самый блок, что и на основной странице сервиса;

- список товаров, реализованный как List, содержащий специфические элементы класса ProductCard (карточки товаров).

Класс ProductCard представляет собой уточнение блока со списком товаров для каждого товара по отдельности и определяет элементы:

- наименование товара;

- цена товара.

Для страницы YmSearchResultsPage также определено действие с помощью аннотации @ActionTitle – проверка присутствия продукта (см. рисунки 2.20, 2.21).

YmSearchResultsPage.java

```
1  package ru.andzhil.pages.YandexMarket;
2
3  import ...
16
17 @PageEntry(title = "Результаты поиска товаров")
18 public class YmSearchResultsPage extends Page {
19
20     private HeaderBlock headerBlock;
21
22     @ElementTitle("Список товаров")
23     @FindBy(xpath = ".//div[contains(@class,'n-snippet-card2 ')]")
24     private List<ProductCard> productCards;
25
26     public YmSearchResultsPage(){
27         PageFactory.initElements(
28                 new HtmlElementDecorator(
29                         new HtmlElementLocatorFactory(
30                                 PageFactory.getDriver())), page: this);
31     }
32
33
34     @ActionTitle("проверяет присутствие продукта")
35     public void compareProductCost(String productName){
36
37         for (ProductCard card: productCards) {
38             if(card.getProductName().toLowerCase().contains(productName.toLowerCase())){
39                 return;
40             }
41         }
42
43         Assert.fail("Продукт " + productName +  " не был найден");
44     }
45 }
```

Рисунок 2.20 Класс страницы с результатами поиска на портале «Яндекс.Маркет»

ProductCard.java

```java
package ru.andzhil.elements.YandexMarket;

import ...

public class ProductCard extends TypifiedElement {

    private final String nameXpath = ".//div[@class='n-snippet-card2__title']/a";
    private final String costXpath = ".//div[@class='price']";

    public ProductCard(WebElement wrappedElement) { super(wrappedElement); }

    public String getProductName(){
        String name = getWrappedElement()
                .findElement(By.xpath(nameXpath))
                .getText();
        return name;
    }

    public Integer getMinCost(){
        String costString = getWrappedElement()
                .findElement(By.xpath(costXpath))
                .getText()
                .replace( oldChar: ' ', Character.MIN_VALUE)
                .replace( target: "руб.", replacement: "")
                .substring(2);

        return Integer.parseInt(costString);
    }
}
```

Рисунок 2.21 Класс элемента конкретного продукта на странице результатов поиска

В итоге, когда классы страниц запрограммированы, остается создать класс для запуска теста, и сам тестовый сценарий на языке Cucumber (см. рисунки 2.22, 2.23).

CucumberTest.java

```
1   package ru.andzhil;
2
3   import ...
6
7   @RunWith(TagCucumber.class)
8   @CucumberOptions(monochrome = true, format = {"pretty"},
9           glue = {"ru.andzhil.autoframework.stepdefs"},
10          features = {"src/test/resources/features/"})
11  public class CucumberTest {}
12
```

Рисунок 2.22 Класс для запуска тестов с использованием Junit

YandexMarket.feature

```
1   # language:ru
2   Функционал: Тестирование поиска Yandex Market
3
4     @ymtest
5     Сценарий: Просмотр результатов поиска
6       * пользователь переходит на страницу "http://market.yandex.ru" по ссылке
7       * открывается страница "Яндекс Маркет"
8       * пользователь в блоке "Меню поиска" (выполняет поиск) с параметром "Пылесосы"
9       * открывается страница "Результаты поиска товаров"
10      * пользователь (проверяет присутствие продукта) "KARCHER WD"
11
```

Рисунок 2.23 Тестовый сценарий (feature) на языке Cucumber

Если все было правильно настроено, то дальше следует запустить класс запуска CucumberTest на выполнение. Он найдет привязанные к нему тестовые сценарии (feature) и запустит их последовательно. В данном случае – будет запущен 1 тестовый сценарий.

По результатам выполнения будет сформирован пакет файлов с результатами для фреймворка Allure 2.0. Данные файлы можно найти в папке «allure-results». Запустить отчет на просмотр можно с помощью средств фреймворка Allure, запустив файл allure.bat, настроив его на файлы с отчетами, либо открыв файл index.html. Внешний вид окна с результатами прогона теста представлен на рисунке 2.24.

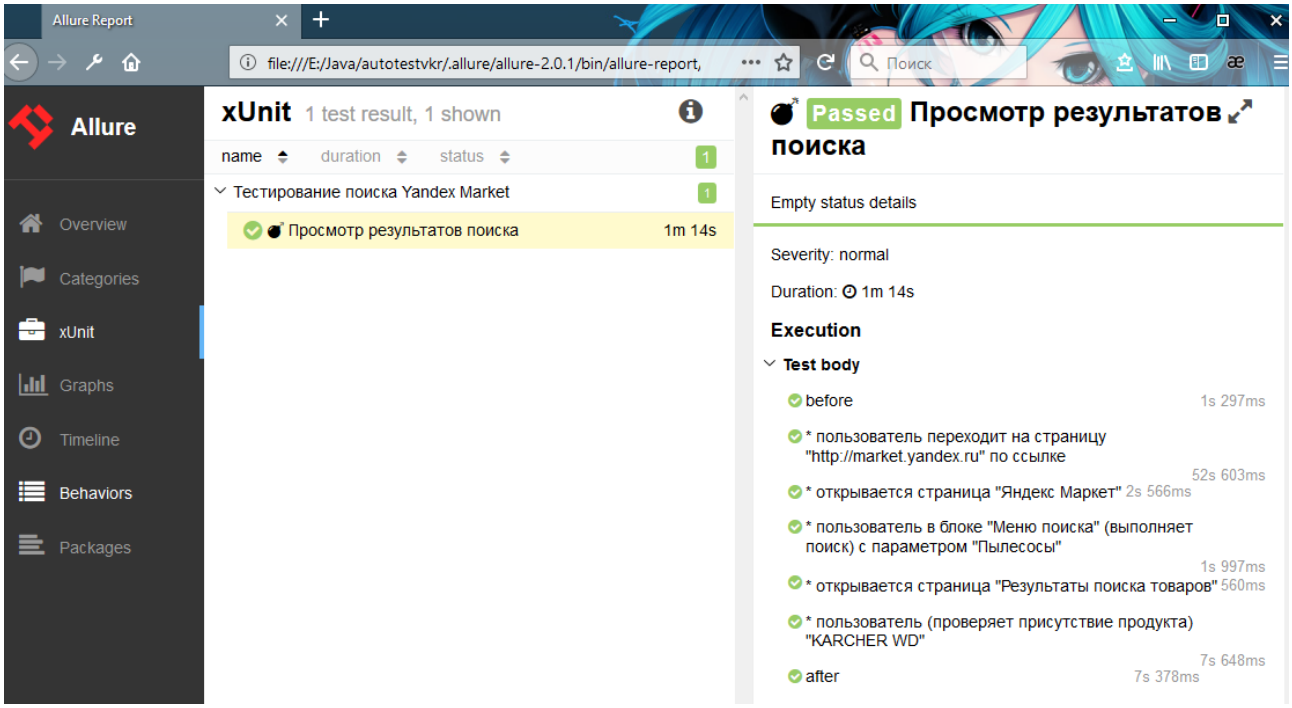

Рисунок 2.24. Просмотр результатов тестов в системе Allure

По данному отчету легко определить, что статус прохождения всего теста «Passed», то есть успешно пройден. В отчете также видно, что все шаги отмечены зелеными значками, то есть все шаги пройдены успешно. В случае падений теста, шаг, на котором тест завершился, и невыполненные шаги будут отмечены особыми цветами, а также будет другой статус у всего теста, например «Failed» (Провален) или «Broken» (Сломан).

Таким, образом, разработав только классы веб-страниц с нужными элементами, и написав тестовый сценарий на человекочитаемом языке, с помощью разработанного фреймворка получилось запустить тест сложного корпоративного портала, такого как «Яндекс.Маркет».

Выводы и рекомендации по главе 2

В главе 2 была описана структура разрабатываемого фреймворка для автоматического тестирования, а также описаны подробности реализованной функциональности.

В отдельном параграфе описана архитектурная составляющая проекта, включающая используемые паттерны, методики, а также средства, инструменты и языки программирования, использованные в проекте. Таким

образом, можно представить архитектуру приложения в целом и оценить применимость разработанного инструмента для определенных проектов.

Была разработана логическая модель и диаграмма классов для проектируемого программного средства, на основе которых была разработана функциональная и структурная схемы проекта.

Были описаны подробно все основные функциональные классы, как с основным функционалом, так и со служебными функциями. Таким образом, была получена спецификация классов по функциональности и подчеркнута универсальность разработанного решения для широкого круга задач по автоматизации тестирования.

В заключение второй главы был пошагово описан контрольный пример реализации системы тестирования, основанной на разработанном фреймворке. Таким образом, практически показано удобство разработки нового теста.

Из всего описанного можно сделать вывод, что фреймворк является готовым к использованию программным средством, при этом за счет открытого исходного кода, готовым к дальнейшему наращиванию функциональности и возможностей.

Глава 3. Оценка и обоснование экономической эффективности проекта

3.1. Выбор и обоснование методики расчета экономической эффективности

С экономической точки зрения внедрение фреймворка автоматизированного тестирования, основанного на принципах BDD, позволит получить высвобождение ресурсов для отдела тестирования, так как автоматические тесты могут прогоняться либо совсем без участия человека, либо с минимальным вмешательством, требуемым для настройки запусков и окружений тестов. Также выигрыш может быть получен за счет увеличения скорости прогона автоматизированных тестов. Плюсом является также высокий уровень повторяемости, то есть обеспечения сравнимых прогонов, исключающих человеческий фактор, то есть ошибки по невнимательности.

Методика расчета экономической эффективности будет включать в себя:

1. Определение длительности работ и календарного плана. Будет проведена оценка времени, необходимого на проектирование, разработку и внедрение фреймворка.
2. Расчет расходов на заработную плату специалистов. Для оценки экономической эффективности необходимо рассчитать планируемые расходы на заработную плату разработчиков, так как в IT компании это является основным ресурсом.
3. Расчет расходов на сопутствующие материалы.
4. Расчет расходов на оборудование.
5. Определение полной себестоимости разработки фреймворка.
6. В итоге, будут рассчитаны показатели прямой эффективности от проекта внедрения фреймворка.

Таким образом, путем расчетов можно будет убедиться в экономической выгоде, получаемой от проекта разработки, и на основе этих данных принять управленческое решение о возможности разработки и внедрения системы.

3.2. Расчет показателей экономической эффективности проекта

3.2.1. Определение длительности работ и календарного плана

Для расчета затрат на этапе проектирования, необходимо, в первую очередь, составить календарный план работ и определить длительность каждой работы. В данной работе для определения длительности работ будет использоваться расчетный метод с использованием экспертных оценок по формуле:

$$t_j^o = \frac{3t_{\min} + 2t_{\max}}{5},$$

где t_j^o - ожидаемая длительность j-й работы, $t_{\min}$ и $t_{\max}$ - наименьшая и наибольшая, по мнению эксперта, длительность работы.

Результаты расчетов сведены в таблицу 3.1.

Таблица 3.1. Длительность этапа разработки

№	Наименование работы	Длительность работы, человекодни		
		$t_{\min}$	$t_{\max}$	t_j^o
1	Разработка технического задания	7	14	10
2	Изучение литературы и документации	7	14	10
3	Анализ методик автоматизированного тестирования веб-приложений	3	7	5
4	Разработка методов решения задачи	15	30	21
5	Классификация тестовых шагов	3	7	5
6	Разработка программных компонентов	15	60	33
7	Тестирование и отладка системы	7	30	17
8	Составление технической документации	1	7	4
9	Составление технико-экономического обоснования	1	7	4
10	Сдача проекта	1	1	1
Итого		60	177	110

Далее, необходимо провести расчеты затрат по конкретным специалистам компании. Результаты расчетов представлены в таблице 3.2.

Таблица 3.2. Расчет трудоемкости проекта по исполнителям

№	Наименование работы	Исполнитель	Трудоемкость, t_j^o, ед.t	Ставка, руб./ ед.t
1	Разработка технического задания	Инженер	7	4000
		Руководитель	3	6000
2	Изучение литературы и документации	Инженер	10	4000
3	Анализ методик автоматизированного тестирования веб-приложений	Инженер	3	4000
		Руководитель	2	6000
4	Разработка методов решения задачи	Инженер	21	4000
5	Классификация тестовых шагов	Инженер	3	4000
		Руководитель	2	6000
6	Разработка программных компонентов	Инженер	33	4000
7	Тестирование и отладка системы	Инженер	17	4000
8	Составление технической документации	Инженер	2	4000
		Руководитель	2	6000
9	Составление технико-экономического обоснования	Руководитель	4	6000
10	Сдача проекта	Инженер	1	4000
Общая трудоемкость		Инженер	97	
		Руководитель	13	

3.2.2. Расчет расходов на заработную плату специалистов

По данным, представленным в предыдущем параграфе 3.2.1 можно рассчитать расходы на заработную плату сотрудников и обязательных отчислений из заработной платы.

Дневную ставку инженера принимаем равной 4000 рублей. Дневную ставку руководителя принимаем равной 6000 рублей.

Расходы на основную заработную плату рассчитаем по формуле:

$$З_{осн.з/пл} = \sum_{i=1}^{k} T_i * C_i,$$

где $З_{осн.з/пл}$ - расходы на основную заработную плату специалистов, k – количество специалистов, T_i – время, потраченное i-м специалистом на проведение работ (дни), C_i – дневная ставка i-го специалиста, рублей/день.

Итого, $З_{осн.з/пл} = 97 * 4000 + 13 * 6000 = 466\ 000$ рублей.

Рассчитаем сумму обязательных отчислений в социальные фонды. Текущий процент отчислений на 2018 год равен 30% (22% - пенсионное страхование, 5,1% - обязательное медицинское страхование, 2,9% - страхование от несчастных случаев).

$З_{соц.страх} = 466\ 000 * 30\% = 139\ 800$ рублей.

Итоговые расходы на оплату труда персонала составят 605 800 рублей.

3.2.3. Расчет расходов на сопутствующие материалы

Проведем расчет потребностей в затратах на сырье и материалы по формуле:

$$З_{материалы} = \sum_{i=1}^{L} G_1 Ц_1 (1 + \frac{Н_{т.з.}}{100}),$$

где $З_{материалы}$ – затраты на сырье и материалы, руб., i – индекс вида сырья и материалов, L – количество видов сырья, G_1 – норма расхода i-го материала (руб./ед.), $Ц_1$- цена приобретения i-го материала (руб./ед.), $Н_{т.з.}$ - норма транспортно-заготовительных расходов (%).

Примем норму транспортно-заготовительных расходов равной 10%.

Результаты расчета нормы материалов сведены в таблицу 3.3.

Таблица 3.3. Затраты на сырье и материалы

Наименование	Тип	Норма расхода, ед.	Цена за единицу, руб.	Общая сумма, руб.
Бумага для принтера	А4	5 шт.	500	2750
Офисные принадлежности	Ручки, карандаши, степлеры	4 шт.	150	660
Картридж для принтера	Картридж Epson	1 шт.	5000	5500
Услуги доступа в Интернет	Широкополосный доступ	2 месяца	6000	13200
Итого				22110

Комплектующие материалы в данном проекте не требуются. Итого, сумма расходов на сырье, материалы и необходимые услуги связи составила 22110 рублей.

3.2.4. Расчет расходов на оборудование

Затраты на содержание и эксплуатацию оборудования определяются из расчета на 1 час работы оборудования с учетом стоимости и производительности оборудования:

$$З_{сэо} = \sum_{i=1}^{m} C_i^{мч} * t_i^{м},$$

где $З_{сэо}$ - затраты на содержание и эксплуатацию оборудования (руб.); $C_i^{мч}$ - расчетная себестоимость одного машино-часа работы оборудования на *i*-й технологической операции (руб./м-ч); $t_i^{м}$ - количество машино-часов, затрачиваемых на выполнение *i*-й технологической операции (м-ч).

Для расчета стоимости машино-часа работы оборудования, необходимо рассчитать годовую стоимость эксплуатации и эффективный фонд рабочего

времени, при этом годовая стоимость включает в себя стоимость электроэнергии, технического обслуживания и амортизационных отчислений.

Стоимость электроэнергии:

$С_{ээ}$= М * $k_з$ * $F_{эф}$ * $С_{кВт.ч}$,

где $М$ – мощность ЭВМ (0,3 кВт); $k_з$ – коэффициент загрузки (0,8); $С_{кВт.ч}$ – стоимость 1 кВт час электроэнергии (4,53руб.); $F_{эф}$ – эффективный фонд рабочего времени, рассчитывается по формуле:

$$F_{эф} = Д_{ном} * d * (1 - \frac{f}{100}),$$

где $Д_{ном}$ = 247 – номинальное число рабочих дней в году; d = 8 – продолжительность рабочего дня, в часах; f = 2% - планируемый процент времени на ремонт ЭВМ.

Эффективный фонд составляет $F_{эф} = 247 * 8 * \left(1 - \frac{2}{100}\right) = 1937$ часов

Стоимость электроэнергии за год составит:

$$С_{ээ} = 0{,}3 * 0{,}8 * 1937 * 4{,}53 = 2\,106 \text{ рублей}$$

Итоговая годовая стоимость электроэнергии для 3 компьютеров составляет 2106 * 3 = 6318 рублей.

Техническое обслуживание и текущий ремонт составляют 2,5% от стоимости оборудования. В нашем случае стоимость компьютеров 3 специалистов составляет: 50 000 руб. * 3 = 150 000 рублей.

Таким образом, затраты на техобслуживание составят 150 000 * 0,025 = 3750 рублей.

Амортизационные отчисления зависят от нормы амортизации. Для целей настоящего расчета примем норму амортизации за 25%.

Амортизационные отчисления за год = 150 000 * 0,25 = 37 500 рублей.

Суммарные годовые эксплуатационные затраты составят:

$С_э$ = $С_{ээ}$ + $С_{то}$ + $А_i$ = 6318 + 3750 + 37500 = 47658 рублей.

Стоимость 1 часа рабочего времени:

$С^{мч}$ = 47658 / 1937 = 24,6 руб.

Общее количество машино-часов, затрачиваемых на выполнение проекта:

$t^{м}$ = 110 * 8 = 880 м-ч

Итого, общие затраты на содержание и эксплуатацию оборудования:

$З_{сэо}$ = 24,6 * 880 = 21 648 рублей.

Процент накладных расходов по данным предприятия – 15%.

Накладные расходы рассчитываются по формуле:

$З_{НР} = З_{ОНС} * 0,15$,

где $З_{НР}$ – сумма накладных расходов, $З_{ОНС}$ – основные расходы на заработную плату.

Знр = 466 000 * 0,15 = 69 900 рублей.

3.2.5. Расчет себестоимости продукта

Для расчета совокупной величины затрат, связанных с выполнением проекта, все приведенные расчеты сводятся в таблицу 3.4.

Таблица 3.4. Себестоимость проекта

№	Наименование статьи затрат	Сумма, руб.
1	Расходы на оплату труда	466 000
2	Отчисления на социальные нужды	139 800
3	Материалы	22 110
4	Расходы на содержание и эксплуатацию оборудования	10 068
5	Амортизационные отчисления	37 500
6	Накладные расходы	69 900
ИТОГО затрат		745 378

Рассчитанная итоговая сумма себестоимости проекта разработки фреймворка автоматизированного тестирования составила 745 378 рублей. При этом основная часть расходов приходится на основную заработную плату инженеров-специалистов.

3.2.6. Расчет показателей итоговой эффективности

Для расчета прямого эффекта от внедрения разработанного фреймворка необходимо рассмотреть показатели трудовых и стоимостных затрат.

Абсолютное снижение трудовых затрат рассчитаем по формуле:

$Т = T_0 - T_1$,

где T_0 – время, затрачиваемое на выполнение автоматизируемых операций в базовом варианте, T_1 – время, затрачиваемое на выполнение автоматизируемых операций в проектном варианте.

Возьмем за основу 8 часов в день, затрачиваемых тестировщиком на ручной прогон тестов. В проектном варианте заложим вместо этого 1 час на подготовку запусков и разбор результатов прогонов тестов, так как на сами прогоны время тестировщика уже не будет тратиться.

Т = 8 – 1 = 7 часов.

Рассчитаем *коэффициент относительного снижения трудовых затрат* $К_т$ (в процентах):

$К_т$ = (7/8) * 100% = 87,5%

Абсолютное снижение стоимостных затрат рассчитаем по формуле:

$С = С_0 - С_1$,

где $С_0$ – стоимостные затраты на обработку информации по базовому варианту, $С_1$ – стоимостные затраты на обработку информации по предлагаемому варианту.

Возьмем за основу 8 часовую заработную плату специалиста по ручному тестированию. В проектном варианте заложим заработную плату за 1 час работы тестировщика – 500 рублей.

С = 4000 – 500 = 3500 рублей.

Рассчитаем *коэффициент относительного снижения стоимостных затрат* $С_т$ (в процентах):

$С_т$ = (3500/4000) * 100% = 87,5%

Помимо рассмотренных показателей целесообразно также рассчитать срок окупаемости затрат на внедрение проекта ($Т_{ок}$):

$$Т_{ок} = \frac{К_п}{\Delta С},$$

где $К_п$ – капитальные затраты на создание проекта.

Срок окупаемости рассчитаем для отдела тестирования, состоящего из 20 специалистов по ручному тестированию:

$Т_{ок}$ = 745 378 / ((3500) * 20) = 10,6 дней.

Интерпретация данного параметра следующая: за 10,6 дней, которые команда ручных тестировщиков потратила бы на проведение вручную автоматизированных тестов, полностью окупается проект по созданию фреймворка автоматизированного тестирования. Дополнительный срок окупаемости зависит от скорости написания новых автоматизированных тестовых сценариев, расчет которой выходит за рамки текущего исследования.

Рассчитаем срок окупаемости для небольшой компании с отделом тестирования, состоящим из 5 специалистов по ручному тестированию:

$Т_{ок}$ = 745 378 / ((3500) * 5) = 42,6 дней.

Данный показатель, также выглядит и является привлекательным, из чего можно сделать вывод о привлекательности разрабатываемого решения для практически любых IT-компаний, имеющих штат специалистов по ручному тестированию.

Выводы и рекомендации по главе 3

В данной главе были представлены расчеты обоснования экономической эффективности проекта. Для этого был рассчитан ряд показателей, таких как полная себестоимость проекта – 745 378 рублей, снижение трудовых и стоимостных затрат – на 87,5%, а также срок окупаемости – от 10,6 до 42,6 дней.

Произведенные расчеты показали эффективность внедрения проектируемого решения, то есть проект можно рекомендовать к реализации и внедрению.

Рассчитанный срок окупаемости – 10,6 дней – довольно низок в случае, если разработанное решение будет применяться в больших отделах тестирования. Но и для небольших компаний срок окупаемости также приемлем – 42,6 дней.

Заключение

Задачей данной выпускной квалификационной работы была разработка фреймворка для автоматизации тестирования веб-приложений. Разработанное решение позволит сократить как время на проведение тестирования, так и количество необходимых для проведения тестирования специалистов, что, несомненно, поспособствует увеличению эффективности компании, которая будет внедрять данную систему в деятельность отдела тестирования программного обеспечения.

В результате работы была создана система, объединяющая компоненты для работы с драйверами для работы с GUI-интерфейсом веб-приложений, для формирования, генерации и хранения тестовых данных, для работы с логами, с печатными формами, а также компонент для создания отчетов по тестированию с помощью внешнего инструмента Allure 2.0.

При разработке были учтены особенности конкретных тестируемых веб-приложений, такие как: работа с конкретными браузерами (версиями браузеров), работа под конкретными операционными системами, для чего был выбран универсальный и расширяемый инструмент Selenium WebDriver. Таким образом, была разработана максимально универсальная система, обеспечивающая максимально удобную возможность расширения под конкретные системы и требования.

В работе был проведен анализ уже существующих фреймворков автоматизированного тестирования, описаны достоинства и недостатки разрабатываемого фреймворка по сравнению с ними. В итоге можно сделать вывод, что разработанное решение имеет достаточное количество достоинств, по сравнению с существующими разработками, что позволяет рекомендовать решение к рассмотрению вопроса о внедрении.

Разработанный фреймворк может быть использован широким кругом как индивидуальных разработчиков программного обеспечения, так и корпоративными пользователями, такими как: отделы/управления тестирования в рамках организационных структур отделов/управлений разработки

программного обеспечения. При этом для более крупных компаний будет наблюдаться более короткий срок окупаемости проекта, что показали расчеты экономической эффективности проекта, представленные в работе.

Апробация решения была произведена на реальных задачах тестирования веб-приложений банковской сферы, что позволило оценить удобство и практическую применяемость разрабатываемого решения.

Дальнейшее развитие программного средства может предполагать расширение имеющегося функционала и внедрение нового, такого как: поддержка новых версий браузеров, разработка дополнительных универсальных шагов, оптимизация/ускорение работы, разработка системы анализа причин выявляемых автотестами ошибок.

Список используемой литературы

1. Бек, К. Шаблоны реализации корпоративных приложений / К. Бек. – М. : Вильямс, 2017. – 176 с.
2. Буч, Г. Язык UML. Руководство пользователя: Пер. с англ. Мухин Н. / Г. Буч, Д. Рамбо, А. Якобсон – М. : ДМК Пресс, 2006. – 496 с.
3. Введение в BDD [Электронный ресурс]. – Режим доступа : http://agilerussia.ru/practices/introducing-bdd/.
4. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств [Текст] : ГОСТ Р ИСО/МЭК 12207-2010; введ. 01.03.12
5. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ. / С. Канер, Д. Фолк, Е. К. Нгуен. – К. : ДиаСофт, 2001. – 544 с.
6. Макконнелл, С. Совершенный код. Мастер класс. Пер. с англ. / С. Макконнелл. – М. : Русская редакция, 2010. — 896 с.
7. Осипенко, Н. Б. Стандартизация и сертификация программного обеспечения. Тексты лекций. / Н. Б. Осипенко. – Министерство образования РБ, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2012. – 155 с.
8. Портал Про Тестинг – Тестирование Программного Обеспечения [Электронный ресурс]. – Режим доступа : http://www.protesting.ru.
9. Тамре, Л. Введение в тестирование программного обеспечения / Л. Тамре. – М. : Вильямс, 2003. – 368 с.
10. Шаблоны корпоративных приложений / М. Фаулер [и др.]. – М. : Вильямс, 2016. – 544 с.
11. Behavior-driven development [Электронный ресурс]. – Режим доступа : https://en.wikipedia.org/wiki/Behavior-driven_development.
12. Guide to the Software Engineering Body of Knowledge [Текст] : SWEBOK, Version 3.0, IEEE; введ. 20.12.13

13. PageObject. Martin Fowler [Электронный ресурс]. – Режим доступа : https://martinfowler.com/bliki/PageObject.html.
14. TIOBE Index [Электронный ресурс]. – Режим доступа : https://tiobe.com/tiobe-index/.

Приложение А

Код фреймворка на CD.