АННОТАЦИЯ

Данная выпускная квалификационная работа посвящена автоматизации тестирования в крупных компаниях. Разрабатывается гибридный фреймворк, позволяющий автоматизировать процессы тестирования людям, не являющимися программистами.

Выпускная квалификационная работа состоит из трех глав.

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

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

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

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

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

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

Задачи выпускной квалификационной работы:

- анализ существующие подходы к тестированию программного обеспечения выявить достоинства и недостатки
- анализ путей автоматизации процесса тестирования
- разработать оптимальный подход к решению проблемы с автоматизации тестирования в крупных компаниях
- разработать архитектуру модулей фреймворка
- реализовать алгоритмы для поиска элементов интерфейса
- составить классификацию и структуру тестовых сценариев
- разработать типовую модель автоматизированных тестов
- подготовить контрольные примеры реализации

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

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

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

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

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

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

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

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

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

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

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

Тестирование программного обеспечения делится на множество подтипов, каждый из которых решает конкретную задачу [1].

Модульное тестирование. Такие тесты пишут разработчики для проработки небольших участков кода. Они позволяют избежать явных

логических ошибок. Модульные тесты обеспечивают покрытием около 80% всего кода, они обязательно включают в себя все методы и логические ветвления.

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

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

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

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

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

Регрессионное тестирование. Подразумевает под собой множество различных тестов, которые прошли успешно в старой версии приложения. Позволяет легко выявлять ошибки, которые появляются в новых версиях приложения. Количество регрессионных тестов увеличивается с каждым новым релизом [1]. Такой подход к тестированию приложения является приемлемым только на небольших объёмах производства программного обеспечения. Чем больше приложение, тем сложнее становится его тестировать и тем больше времени уходит на его выпуск заказчику.

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

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

В современных системах целевым подходом к разработке больших приложений является разбиение его на модули, которые выполняют

небольшую часть конечной функциональности. Такие модули принято называть «микросервисы», а архитектура таких приложений называется «микросервисная архитектура».

Достоинства:

- низкие затраты на тестирование.

Недостатки:

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

Решение:

- увеличить команду тестирования.

Достоинства:

- тестирование проходит быстро.
- нагрузка на каждого тестировщика распределена равномерно.

Недостатки:

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

Решение:

- внедрить автоматизированное тестирование.

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

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

Итого мы имеем множество однообразных действий, которые выполняются человеком в ручном режиме. Такие процессы в реальной жизни принято автоматизировать, тестирование не стало исключением.

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

Рис. 6 - Схема взаимодействия разработчика и автотестера №1

Достоинства:

- автоматизированные тесты работают быстрее любого человека.
- есть возможность запускать тесты в обособленной среде.
- можно быстро выполнять полное тестирование любых версий приложений.
- тесты работают одновременно, в асинхронном режиме.
- тесты не имеют человеческого фактора - безошибочны.
- автоматизировать можно любой процесс, даже отправку отчёта об ошибках разработчикам напрямую.
- тесты легко поддерживать, есть возможность гибкой настройки.

Недостатки:

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

Решение:

- нанять ещё нескольких специалистов по автоматизированному тестированию.

Достоинства:

- автоматизированные тесты разрабатываются и поддерживаются на высокой скорости.
- команда по автоматизации тестирования способна надёжно протестировать практически любой крупный проект

Недостатки:

- большие затраты на тестирование
- приложение практически не проверяется в ручном режиме, из-за этого можно получить ошибки, которые не были предусмотрены автотестом.

Решение:

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

Предлагается разработать приложение из трёх компонентов:

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

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

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

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

Целевое решение имеет ряд преимуществ, таких как:

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

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

Модуль с тестовыми сценариями подразумевает под собой набор тестов, написанных для всех предыдущих релизов, образующих тесты типа регрессионные и набор для нового релиза.

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

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

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

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

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

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

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

В общем виде структуру функционального модуля можно представить следующим образом.

Абстрактный класс BasePage представляют собой класс, в котором заложена общая функциональность, которая необходима каждому дочернему компоненту.

В свою очередь дочерние элементы представляют собой уже конкретные уникальные реализации специфичного тестового сценария, которые встречаются исключительно в нём.

Более подробно реализация будет рассмотрена далее.

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

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

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

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

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

Квадратные скобки означают порядковый номер вложенного элемента.

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

Алгоритм реализован следующим образом.

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

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

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

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

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

Сценарий может состоять из следующих блоков:

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

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

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

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

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

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

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

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

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

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

Для этого создадим специальные директории, в которые разместим файлы с тестовыми сценариями.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Тестируемое приложение имеет следующий функционал:

- авторизация пользователя,
- просмотр продукта,
- выбор товара из каталога,
- наполнение корзины с продуктами,
- оформление заказа.

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

Составим простой сценарий для проверки возможности оформления заказа с одним товаром:

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

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

Реализация сценария подразумевает под собой реализацию каждого шага, описанного в сценарии. Например, реализуем авторизацию на данной странице.

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

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

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

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

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

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

Здесь добавляется общий метод по нажатию кнопки подтверждения, его реализация содержится в классе BaseSteps.

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

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

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

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

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

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

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

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

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

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

– научно обоснованная организация проведения всех этапов экспертизы, обеспечивающая наибольшую эффективность работы на каждом из этапов,

– применение количественных методов как при организации экспертизы, так и при оценке суждений экспертов и формальной групповой обработке результатов.

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

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

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

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

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

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

Таблица 3.1

В качестве экспертов выступили:

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

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

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

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

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

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

Для оценки можно использовать три ранга:

– ранг «1» присуждается самому значимому критерию среди других

– ранг «3» присуждается критерию, имеющему наименьшее значение среди других

– эксперт может присуждать нескольким критериям одинаковые ранги

– каждый эксперт обособлен от других экспертов

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

Таблица 3.2

Проведем обработку результатов ранжирования критериев с целью поиска коэффициентов важности. Если ранги у какого-либо критерия совпадают, необходимо привести ранжировки к нормализованному виду (в соответствие ставится ранг, вычисляемый путем поиска среднего значения номеров мест критериев в ранжировке). Так как в строках Таблицы 3.2. имеются совпадающие значения рангов (например, строки 2, 3, 4, 5), то приводим оценки экспертов к нормальному виду в Таблице 3.3.

Норм. ранжирование	№ критерия			Сумма
	1	2	3	
Эксперт 1	3	2	1	6
Эксперт 2	1,5	3	1,5	6
Эксперт 3	1,5	1,5	3	6
Эксперт 4	2,5	1	2,5	6
Эксперт 5	1,5	3	1,5	6
$X_j = \sum_{i=1}^{m} X_{ij}$	10	10,5	9,5	30

Таблица 3.3

Для проверки правильности нормализации ранжировок вычислим сумму рангов для i-го эксперта по формуле

(m – количество экспертов (5), n - количество критериев (3)):

$$\sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$$

$$\sum_{i=1}^{3} i = \frac{1}{2}3(3+1) = 6$$

В таблице 3.2. сумма ранжировок в каждой строке равна 6 и сходится с полученной в результате расчетов, следовательно, можно сделать вывод о том, что нормализация ранжировок выполнена верно.

Оценим важность каждого критерия для экспертов по формуле:

$$\beta_i = \frac{m*(n+1)- X_j}{\frac{1}{2}*mn(n+1)}$$

(m – количество экспертов (5), n - количество критериев (3))

$$\beta_1 = \frac{5*(3+1)- 10}{\frac{1}{2}*5*3(3+1)} = 0{,}333$$

$$\beta_2 = \frac{5*(3+1)- 10{,}5}{\frac{1}{2}*5*3(3+1)} = 0{,}316$$

$$\beta_3 = \frac{5*(3+1)- 9{,}5}{\frac{1}{2}*5*3(3+1)} = 0{,}35$$

Из расчетов, приведенных выше, следует, что для данной группы экспертов порядок важности критериев следующий:

1. Поддержка (оценка того, насколько просто будет поддерживать фреймворк)
2. Доступность (оценка простоты использования фреймворка)
3. Информативность отчётов (оценка того, насколько информативными получаются отчёты о тестировании)

Далее экспертам было предложено оценить функциональность приложения с использованием бальной системы. Оценки от пяти экспертов приведены в Таблице 3.4.

Обозначения оценок:

5 – «Отлично»

4 – «Хорошо»

3 – «Допустимо»

2 – «Плохо»

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

$$x = \sum_{j=1}^{n} X_j * \beta_j = (4{,}8 * 0{,}333) + (4{,}6 * 0{,}316) + (4{,}6 * 0{,}35) = 4{,}667$$

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

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

Разработанная система имеет широкий функционал, позволяет проводить автоматизированное тестирование.