****Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

**«ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ УПРАВЛЕНИЯ»**

Институт Информационных систем

Кафедра Программной инженерии

**Выпускная квалификационная (бакалаврская) работа**

Обучающегося Алексеева Федора Матвеевича

(Фамилия, Имя, Отчество)

на тему «Автоматизация функционального тестирования веб-приложений»

***Направление подготовки*** 09.03.03 – «Прикладная информатика»

(шифр) (название)

***Направленность (профиль) подготовки*** Прикладная информатика

(название)

**ОБУЧАЮЩИЙСЯ** Алексеев Ф. М.

(Фамилия, Инициалы)

**РУКОВОДИТЕЛЬ** к.т.н., доцент, Макаренко А.Е.

(ученая степень, звание, Фамилия и Инициалы)

**КОНСУЛЬТАНТ** к.т.н., доцент, Макаренко А.Е.

(ученая степень, звание, Фамилия и Инициалы)

Допустить к защите

**ЗАВЕДУЮЩИЙ КАФЕДРОЙ** к.т.н., доцент, Терелянский П. В.

(ученая степень, звание, Фамилия и Инициалы)

" \_\_\_ " \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**Москва – 2018**

Оглавление

[Введение 3](#_Toc513194852)

[Глава 1. Теоретические аспекты процесса тестирования 6](#_Toc513194853)

[1.1. Определение понятия тестирования ПО 6](#_Toc513194854)

[1.2. Классификация видов тестирования 7](#_Toc513194855)

[1.3. Методологии тестирования 17](#_Toc513194856)

[1.4. Процесс тестирования 18](#_Toc513194857)

[1.4.1. Разработка тест-кейсов 19](#_Toc513194858)

[1.4.2. Выполнение тест-кейсов 23](#_Toc513194859)

[1.4.3. Анализ результатов тестирования 24](#_Toc513194860)

[1.5. Описание процесса автоматизированного тестирования 25](#_Toc513194861)

[1.6. Критерии эффективности процесса автоматизированного тестирования 28](#_Toc513194862)

[Глава 2. Автоматизация процесса тестирования 32](#_Toc513194863)

[2.1. Описание компании 32](#_Toc513194864)

[2.2. Расчёт экономической целесообразности введения автоматизированного тестирования 32](#_Toc513194865)

[2.3. Внедрение автоматизированных тестов 35](#_Toc513194866)

[Заключение 49](#_Toc513194867)

[Список литературы 51](#_Toc513194868)

# Введение

Существенный рост заинтересованности в тестировании программного обеспечения пришелся на девяностые года в США. Быстрое развитие информационных и сетевых технологий привело к увеличению производства на рынке программного обеспечения. Увеличение конкуренции между производителями программного обеспечения требовало повышенного внимания к качеству продукции. Поскольку ассортимент продукции сильно расширился, а цены стали доступнее, потребители начали обращать большее внимание на качество программного обеспечения.

В настоящее время практически все области жизни подвержены компьютеризации. Мало того, что компьютеры используются в повседневной жизни для обычных целей: коммуникаций, покупок товаров, работы, обучения; они также необходимы, когда речь идет о гораздо более значимых сферах, таких как медицина, транспорт, строительство, банковская деятельность и многие другие. Таким образом, вопрос о качестве программного обеспечения становится особенно важным, поскольку это не только вопрос комфорта, но и безопасности.

Осознавая вышесказанное, многие компании по всему миру начали инвестировать средства в повышение качества программного обеспечения – начали создаваться отделы и департаменты тестирования и контроля качества, а также применять новые технологии, которые позволили компаниям увеличить свое преимущество на рынке за счет повышения качества создаваемых ими программных продуктов.

В наши дни тестирование программного обеспечения является неотъемлемой частью жизненного цикла разработки программного обеспечения.

Тестирование необходимо для того, чтобы понять, работает ли программа как ожидается и соответствует ли она предъявляемым к ней требованиям. Своевременное выявление и исправление ошибок и недоработок имеет огромное значение в процессе разработки программного продукта, поскольку это уменьшает риски и при этом происходит снижение затрат на разработку программного обеспечения. При недостаточном внимании к качеству ПО, снижается конкурентоспособность продукта, исправление ошибок на стадии эксплуатации приводит к большим финансовым потерям, нежели на стадии разработки. Также имеют место быть репутационные риски и снижение доверия клиентов к продукту. Благодаря тестированию, компании способны поддерживать качество своих продуктов на очень высоком уровне. Процесс тестирования ПО может быть автоматизирован, что во многих случаях положительно отразится на скорости и качестве тестирования. Это позволяет ещё больше снизить издержки компании и повысить качество продукта.

В настоящий момент пристальное внимание уделяется технологиям тестирования, способам минимизировать издержки и автоматизировать процесс тестирования. По данной проблематике существует достаточно большое количество книг и статей, касающихся как общих понятий в сфере тестирования, так и исследований узкой направленности. Целью данного исследования является анализ эффективности использования автоматизированного тестирования. Для достижения поставленной цели были определены следующие задачи:

* Выявление теоретических основ тестирования, классификация и описание его видов.
* Анализ и описание процесса тестирования, выявление критериев корректно построенного процесса.
* Определение критериев эффективности процесса тестирования.
* Расчёт экономической целесообразности введения автоматизированного тестирования.
* Написание программной основы с реализованными функциональными тестами двух видов: **API** (application programming interface) – тесты, **GUI** (graphical user interface) – тесты.

Для данного исследования была использована литература, описывающая различные направления тестирования. Для решения теоретических задач, таких как анализ процессов тестирования и его видов, использовались фундаментальные, классические для области тестирования книги, такие как «Ключевые процессы тестирования» автора Рекс Блэк, «Software Testing» автора Ron Patton. Для решения задач, связанных с автоматизированным тестированием, использовалась литература более узкой направленности, такая как «Автоматизация процессов тестирования» Винниченко И. В. При изучении экономических аспектов автоматизированного тестирования и расчетах использовалась следующие книги и статьи: Александр Хрущев «Эффективность использования автоматических тестов в ИТ-проектах», Максим Черняк «Оценка эффективности автоматизации тестирования» и «Автоматизированное тестирование программного обеспечения» авторов Элфрид Дастин, Джефф Рэшка, Джон Пол ([список литературы](#_Список_литературы)).

Структура работы обусловлена целью и задачами исследования. Работа состоит из введения, двух глав и заключения.

В введении отражена актуальность работы, описаны цель, задачи и методы исследования, дана характеристика основных источников информации.

В первой главе рассматривается понятие тестирования, классифицируются виды тестирования, описываются методологии тестирования. Также описан процесс тестирования и выделены моменты, которые требуют более пристального внимания при выполнении тестирования. Здесь же определены критерии и показатели эффективности тестирования, раскрыто понятие автоматизированного тестирования, описаны случаи, когда автоматизированное тестирование целесообразно, и проанализированы достоинства и недостатки автоматизированного тестирования.

Вторая глава посвящена написанию программной основы, или фреймворку, с реализованными автоматизированными тестами двух типов: API – тесты, GUI – тесты. В ней рассматривается целесообразность автоматизированного тестирования, обосновывается выбор инструмента реализации, приведены написанные автоматизированные тесты.

Заключение состоит из итогов исследования и окончательных выводов по рассматриваемой теме, сформированных в процессе данного исследования.

В приложении предоставлен полный исходный код решения по автоматизации тестов.

**Глава 1. Теоретические аспекты процесса тестирования**

В данной главе описаны такие вопросы, как выявление теоретических основ тестирования, классификация и описание видов тестирования, анализ и описание процесса тестирования, выявление критериев корректно построенного процесса.

Это необходимо для того, чтобы лучше понимать процессы тестирования, различать виды тестирования и применять знания при оценке целесообразности внедрения автоматизированного тестирования в компании.

* 1. **Определение понятия тестирования ПО**

**Тестирование** – это одна из техник контроля качества, которая включает в себя такие процессы, как проектирование тестов, выполнение тестирования и анализ полученных результатов.

**Тестирование программного обеспечения** – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

*Рисунок 1. Общая схема тестирования.*

На входе тестировщик получает программу, которую необходимо тестировать, и предъявляемые к ней требования. Наблюдая за программой в определенных условиях, на выходе тестировщик получает информацию о соответствии или несоответствии программы требованиям. Данная информация используется для исправления ошибок в существующем продукте, любо для изменения требований к еще только разрабатываемому продукту.

Тест (проверка) включает в себя выбранную определенным образом искусственно созданную ситуацию и описание наблюдений, которые нужно осуществить, для проверки программы на соответствие определенным требованиям. Самый простой пример, авторизация на странице интернет банка, после которой пользователю долдн6а отобразиться главная страница.

Тест может быть как коротким и проводиться за несколько минут, так и длинным и занимать целые часы, например, тест производительности, проверяющий работоспособность системы при длительной нагрузке.

Таким образом, в процессе тестирования тестировщик выполняет две основные задачи.

Первой задачей является управление выполнением программы, а также создание искусственных ситуаций, в которых и происходит проверка поведения программы.

Вторая задача состоит в наблюдении за тем, как программа ведет себя в различных созданных ситуациях, и в сравнении того, что видит тестировщик, с тем, что ожидается.

Если рассматривать задачи современного тестирования, то можно прийти к заключению, что они заключаются не только в обнаружении ошибок в программах, но и в выявлении причин, по которым эти ошибки возникают. Такой подход к процессу тестирования позволяет разработчикам выполнять свою работу с максимальной эффективностью, устраняя обнаруженные ошибки быстро и своевременно.

* 1. **Классификация видов тестирования**

При тестировании программного продукта применяется огромное количество различных видов тестов. Наиболее широкую и подробную классификацию предложил автор книги «Тестирование Дот Ком» Роман Савин. Он объединил виды тестирования по таким признакам, как объект, субъект тестирования, уровень, позитивность тестирования, и степень автоматизации тестирования. Классификация была дополнена на основании таких источников, как книга Сэма Канера, «Тестирование программного обеспечения» и интернет-ресурс, посвященный тестированию, «Про Тестинг - Тестирование Программного Обеспечения» ([список литературы](#_Список_литературы)).

**По объекту тестирования**

* **Нефункциональное тестирование.** Позволяет проверить соответствие свойств программного обеспечения поставленным нефункциональным требованиям. Таким образом, нефункциональное тестирование - это тестирование всех свойств программы, не относящихся к функциональности системы. Такими свойствами могут быть предъявленные характеристики с точки зрения таких параметров как:
  + Надежность (способность системы реагировать на непредвиденные ситуации. Например, выключение одного из двух серверов приложения).
  + Производительность (способность системы работать под большими нагрузками. Например, зарплатный день в интернет банке).
  + Удобство (исследование удобства работы пользователя с приложением).
  + Масштабируемость (возможность масштабировать приложение как вертикально, так и горизонтально. Вертикальное масштабирование — увеличение производительности каждого компонента системы с целью повышения общей производительности. Горизонтальное масштабирование — разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам (или их группам), и (или) увеличение количества серверов, параллельно выполняющих одну и ту же функцию).
  + Безопасность (исследование возможности нарушения работы приложения и кражи пользовательских данных злоумышленниками).
  + Портируемость (возможность перенести приложение на определенный набор платформ)

И много других качеств.

* **Функциональное тестирование.** Функциональное тестирование на сегодняшний день является одним из наиболее часто применяемых видов тестирования. Задача такого тестирования - это установить, насколько соответствует разработанное программное обеспечение (ПО) требованиям заказчика с точки зрения функционала. Иначе говоря, проведение функциональных тестов позволяет проверить способность информационной системы решать задачи пользователей.
* **Тестирование пользовательского интерфейса.** Это тестирование корректности отображения элементов пользовательского интерфейса на различных устройствах, правильности реагирования их на совершение пользователем различных действий и оценка того, насколько ожидаемо ведет себя программа в целом. Такое тестирование дает возможность оценить, насколько эффективно пользователь сможет работать с приложением и насколько внешний вид приложения соответствует стандартам и требованиям, заявленным и утвержденным заказчиком. При проведении тестирования пользовательского интерфейса основной задачей тестировщика является выявление визуальных и структурных недостатков в графическом интерфейсе приложения, проверке возможности и удобства навигации в приложении и корректность обработки приложением ввода данных с клавиатуры, мыши и других устройств ввода.
* **Тестирование удобства использования.** Это способ тестирования, позволяющий оценить степень удобства использования приложения, скорость обучения пользователей при работе с программой, а также насколько пользователи разрабатываемого продукта находят ее понятной и привлекательной в контексте заданных условий. Такое тестирование необходимо для обеспечения максимально положительного пользовательского опыта при работе с приложением.
* **Тестирование защищенности.** Позволяет выявить главные уязвимости программного обеспечения по отношению к различным атакам со стороны злоумышленников. Компьютерные системы довольно часто подвергаются кибератакам с целью нарушения работоспособности информационной системы либо кражи конфиденциальных данных. Тестирование безопасности дает возможность проанализировать реальную реакцию и действенность защитных механизмов, использованных в системе, при попытке проникновения. В процессе тестирования безопасности тестировщик пытается выполнять те же действия, которые выполнял бы настоящий взломщик. При попытке тестировщиком взломать систему могут использоваться любые средства: атаки системы при помощи специальных утилит; попытки узнать логины и пароли с помощью внешних средств; DDOS атаки на вычислительную систему с целью довести её до отказа, то есть создание таких условий, при которых пользователи системы не могут получить доступ к предоставляемым системным ресурсам; целенаправленная генерация ошибок для обнаружения возможности проникновения в систему в процессе её восстановления;
* **Инсталляционное тестирование.** Под этим термином подразумевают тестирование корректности установки (инсталляции) определенного программного продукта. Такое тестирование обычно происходит в искусственно созданных средах с целью выявить степень готовности программного обеспечения к эксплуатации. Основные причины проведения таких тестов связаны с необходимостью проверить корректность поведения программного продукта при автоматизированном развертывании либо обновлении. Обеспечение правильной и стабильной установки программного обеспечения является очень важным фактором при создании программного продукта, поскольку позволяет пользователям быстрее и с меньшими усилиями начать использовать продукт, при этом обеспечивая одинаково корректное поведение этого продукта во всех протестированных программных средах.
* **Конфигурационное тестирование.** Конфигурационное тестирование предназначено для оценки работоспособности программного обеспечения при разнообразных конфигурациях системы. В зависимости от типа тестируемого программного продукта, конфигурационное тестирование может преследовать разные цели. Обычно это либо определение оптимальной конфигурации оборудования, обеспечивающего достаточные для работы ПО параметры производительности, либо проверка определенной конфигурации оборудования (или платформы, включающей в себя помимо оборудования, стороннее ПО, необходимое для работы программы) на совместимость с тестируемым продуктом. Если речь идет о клиент-серверном программном обеспечении, то конфигурационное тестирование проводится отдельно для сервера и отдельно для клиента. Обычно при тестировании совместимости сервера с определенной конфигурацией стоит задача найти оптимальную конфигурацию, поскольку важна стабильность работы и производительность сервера. В то время как при тестировании клиента, наоборот, пытаются выявить недостатки ПО при любых конфигурациях и по возможности устранить их.
* **Тестирование надежности и восстановления после сбоев (стрессовое тестирование).** Такой вид тестирования довольно часто проводится для программного обеспечения, работающего с ценными пользовательскими данными, бесперебойность работы и скорость восстановления после сбоев которого критичны для пользователя. Тестирование на отказ и восстановление осуществляет проверку способности программы быстро и успешно восстанавливаться после отказа оборудования, перебоев сети или критических ошибок в самом программном обеспечении. Это дает возможность оценить возможные последствия отказа и время, необходимое для последующего восстановления системы. На основе полученных в ходе тестирования данных может быть оценена надежность системы в целом, и, при условии неудовлетворительных показателей, соответствующие меры, направленные на улучшение систем восстановления, могут быть приняты.

* **Тестирование локализации.** Тестирование локализации дает возможность выяснить, насколько хорошо приспособлен продукт для населения определенных стран и насколько он соответствует ее культурным особенностям. Обычно, рассматриваются культурный и языковой нюансы, а именно перевод пользовательского интерфейса, сопутствующей документации и файлов на определенный язык, также тестируется правильность форматов валют, чисел, времени и телефонных номеров.
* **Нагрузочное тестирование.** Нагрузочное тестирование позволяет выявить максимальное количество однотипных задач, которые программа может выполнять параллельно. Самая популярная цель нагрузочного тестирования в контексте клиент-серверных приложений - это оценить максимальное количество пользователей, которые смогут одновременно пользоваться услугами приложения.
* **Тестирование стабильности.** Тестирование стабильности проверяет работоспособность приложения при длительном использовании на средних нагрузках. В зависимости от типа приложения, формируются определенные требования к длительности его бесперебойной работы. Тестирование стабильности стремится выявить такие недочеты приложения как утечки памяти, наличие ярко выраженных скачков нагрузки и прочие факторы, способные помешать работе приложения в течение длительного периода времени.
* **Объемное тестирование.** Задачей объемного тестирования поставлено выявление реакции приложения и оценка возможных ухудшений в работе ПО при значительном увеличении количества данных в базе данных приложения. Обычно в такое тестирование входит:
* Замер времени выполнения операций, связанных с получением или изменением данных БД при определенной интенсивности запросов.
* Выявление зависимости увеличения времени операций от объема данных в БД.
* Определение максимального количества пользователей, которые имеют возможность одновременно работать с приложением без заметных задержек со стороны БД.
* **Тестирование масштабируемости.** Это вид тестирования программного обеспечения, предназначенный для проверки способности продукта к увеличению (иногда к уменьшению) масштабов определенных нефункциональных возможностей. Некоторые виды приложений должны легко масштабироваться, и при этом, разумеется, оставаться работоспособными и выдерживать определенную пользовательскую нагрузку.

**Тестирование, связанное с изменениями в ПО.**

* **Санити или санитарное тестирование** – является одним из видов тестирования, целью которого служит доказательство работоспособности конкретной функции или модуля в соответствии с техническими требованиями, заявленными заказчиком. Санитарное тестирование довольно часто используется при проверке какой-то части программы или приложения при внесении в нее определенных изменений со стороны факторов окружающей среды. Данный вид тестирования обычно выполняется в ручном режиме.
* **Дымовое тестирование** представляет собой короткий цикл тестов, целью которых является подтверждение факта запуска и выполнения функций устанавливаемого приложения после того, как новый или редактируемый код прошел сборку. По завершении тестирования наиболее важных сегментов приложения предоставляется объективная информация о присутствии или отсутствии дефектов в работе тестируемых сегментов. По результатам дымового тестирования принимается решение об отправке приложения на доработку или о необходимости его последующего полного тестирования.
* **Регрессионное тестирование** – тестирование, направленное на обнаружение ошибок в уже протестированных участках. Регрессионное тестирование проверяет продукт на ошибки, которые могли появиться в результате добавления нового участка программы или исправления других ошибок. Цель данного вида тестирования – убедиться, что обновление сборки или исправление ошибок не повлекло за собой возникновения новых багов.

**По уровню тестирования**

* **Модульное тестирование (Unit тесты)**. Заключается в проверке каждого отдельного модуля (самобытного элемента системы) путем запуска автоматизированных тестов в искусственной среде. Реализации таких тестов часто используют различные заглушки и драйверы для имитации работы реальной системы. Модульное автоматизированное тестирование - это самая первая возможность запустить и проверить исходный код. Создание Unit тестов для всех модулей системы позволяет очень быстро выявлять ошибки в коде.
* **Интеграционное тестирование.** Это тестирование отдельных модулей системы на предмет корректного взаимодействия. Основная цель интеграционного тестирования - найти дефекты и выявить некорректное поведение, связанное с ошибками в интерпретации или реализации взаимодействия между модулями.
* **Системное тестирование.** Это тестирование программы в целом, такое тестирование проверяет соответствие программы заявленным требованиям.
* **Приемочное тестирование.** Это комплексное тестирование, определяющее фактический уровень готовности системы к эксплуатации конечными пользователями. Тестирование проводится на основании набора тестовых сценариев, покрывающих основные бизнес-операции системы.

**По исполнению кода**

* **Статическое тестирование.** Это выявление артефактов, появляющихся в процессе разработки программного продукта, путем анализа исходных файлов, таких как документация или программный код. НапримерТакое тестирование проводится без непосредственного запуска кода, качество исходных файлов и их соответствие требованиям оцениваются вручную. Статическое тестирование должно проводиться до динамического тестирования, таким образом, ошибки, обнаруженные на этапе статического тестирования, обойдутся дешевле. С точки зрения исходного кода, статическое тестирование выражается в ревизии кода. Обычно ревизия кода отдельных файлов производится после каждого изменения этих файлов программистом, сама же ревизия может проводиться как другим программистом, так и ведущим разработчиком, либо отдельным работником, занимающимся ревизией кода. Использование статического тестирования дает возможность поддерживать качество программного обеспечения на всех стадиях разработки и уменьшает время разработки продукта.
* **Динамическое тестирование.** В отличии от статического тестирования, такой вид тестирования предполагает запуск исходного кода приложения. Таким образом, динамическое тестирование содержит в себе множество других типов тестирования, которые уже были описаны выше. Динамическое тестирование позволяет выявить ошибки в поведении программы с помощью анализа результатов ее выполнения. Получается, что почти все существующие типы тестирования соответствуют классу динамического тестирования.

**По субъекту тестирования**

* **Альфа-тестирование.** Это тестирование проводится для самых ранних версий компьютерного программного обеспечения (или аппаратного устройства). Альфа-тестирование почти всегда проводится самими разработчиками ПО. В процессе альфа-тестирования разработчики приложения находят и исправляют ошибки и проблемы, имеющиеся в программе. Обычно во время Альфа-тестирования происходит имитация работы с программой штатными разработчиками, реже имеет место реальная работа как потенциальных пользователей, так и заказчиков с продуктом. Как правило, альфа-тестирование проводится на самом раннем этапе разработки ПО, однако в отдельных случаях может быть применено для законченного или близкого к завершению продукта, например, в качестве приёмочного тестирования.
* **Бета-тестирование.** Тестирование продукта, по-прежнему находящегося в стадии разработки. При бета-тестировании этот продукт предоставляется для эксплуатации некоторому количеству пользователей для выявления возникающих проблем и недочетов при использовании программного продукта. Такое тестирование необходимо чтобы найти ошибки, которые разработчики могли пропустить. Обычно бета-тестирование проводится в две фазы: закрытый бета-тест и открытое бета-тестирование. Закрытый бета-тест - это тестирование на строго ограниченном кругу избранных пользователей. Такими пользователями могут выступать знакомые разработчиков, либо их коллеги, не связанные напрямую с разработкой тестируемого продукта. Открытое бета-тестирование заключается в создании и размещении в открытом доступе публичной бета-версии. В данном случае любой пользователь может выступать бета-тестером. Обратная связь от таких бета-тестеров осуществляется с помощью отзывов на сайте и встроенных в программу систем аналитики и логирования пользовательских действий, эти системы необходимы для анализа поведения пользователей и обнаружения трудностей и ошибок, с которыми они сталкиваются.

**По позитивности сценария**

* **Позитивное тестирование.** Тесты с позитивным сценарием проверяют способность программы выполнять заложенный в нее функционал. Как правило, для такого тестирования разрабатываются тестовые сценарии, при выполнении которых, в нормальных для ПО условиях работы, не должно возникать никаких сложностей.
* **Негативное тестирование.** Негативное тестирование программного обеспечения происходит на сценариях, соответствующих нештатному поведению программы. Такие тесты проверяют корректность работы программы в экстренных ситуациях. Это позволяет удостовериться в том, что программа выдает правильные сообщения об ошибках, не повреждает пользовательские данные и ведет себя корректно в целом при ситуациях, в которых не предусмотрено штатное поведение продукта. Основная цель негативного тестирования - это проверить устойчивость системы к различным воздействиям, способность правильно валидировать входные данные и обрабатывать исключительные ситуации, возникающие как в самих программных алгоритмах, так и в бизнес-логике.

**По степени автоматизации**

* **Ручное тестирование.** Ручное тестирование проводится без использования дополнительных программных средств, оно позволяет проверить программу или сайт с помощью имитации действий пользователя. В этой модели тестировщик выступает в качестве пользователя, следуя определенным сценариям, параллельно анализируя вывод программы и ее поведение в целом.
* **Автоматизированное тестирование.** Такое тестирование позволяет за счет использования дополнительного программного обеспечения для автоматизации тестов значительно ускорить процесс тестирования. Такое дополнительное ПО позволяет контролировать и управлять выполнением тестов и сравнивать ожидаемый и фактический результаты работы программы. Более подробно оно будет рассмотрено позже.
  1. **Методологии тестирования**

Существуют различные методологии динамического тестирования ПО. В зависимости от наличия у тестировщика доступа к исходному коду программы, выделяют следующие методы тестирования:

* Метод черного ящика
* Метод белого ящика
* Метод серого ящика

**Метод черного ящика.**

Впервые термин «черный ящик» упоминается психиатром У. Р. Эшби в книге "Введение в кибернетику" в 1959 г. Он писал, что метод черного ящика позволяет изучать поведение системы, абстрагируясь от ее внутреннего устройства.

В области тестирования метод черного ящика - это техника тестирования, которая основана на работе с внешними интерфейсами программного обеспечения, без знания внутреннего устройства системы.

Данный метод назван “Черным ящиком”, поскольку в этом методе тестируемое программное обеспечение для тестировщика выглядит как черный ящик, внутри которого происходят некоторые процессы, однако тестировщику о них принципиально ничего не известно. Данная техника позволяет обнаружить ошибки в следующих категориях:

* Ошибки интерфейса.
* Недостающие или неправильно реализованные функции.
* Недостаточная производительность или ошибки поведения системы.

Таким образом, поскольку тестировщик не имеет никакого представления о внутреннем устройстве и структуре системы, ему необходимо сконцентрироваться на том, что делает программа, а не на том, как она это делает.

**Метод белого ящика.**

Как можно догадаться из названия, этот метод тестирования противоположен методу черного ящика. Данный метод тестирования основан на анализе внутренней структуры системы.

То есть в данном случае тестировщику известны все аспекты реализации тестируемого программного обеспечения. Этот метод позволяет протестировать не только корректность реакции программы на определенный ввод (как в случае с черным ящиком), но и правильную работу отдельных модулей и функций, основываясь на знании кода, который будет обрабатывать этот ввод. Знание особенностей реализации тестируемой программы – обязательное требование к тестировщику для успешного применения этой техники. Тестирование методом белого ящика позволяет углубиться во внутренне устройство ПО, за пределы его внешних интерфейсов.

**Метод серого ящика.**

Этот метод тестирования системы предполагает комбинацию подходов Белого и Черного ящиков. Таким образом, тестировщику лишь частично известно внутреннее устройство программы. Например, предполагается, наличие доступа к внутренней структуре программного обеспечения для разработки максимально эффективных тест-кейсов, в то время как само тестирование будет проводиться методом черного ящика. Или тестировщики могут во всем следовать методу черного ящика, однако для того, чтобы убедиться в корректной работе отдельных алгоритмов, могут смотреть информацию в логах или анализировать записи программы в базе данных.

* 1. **Процесс тестирования**

Тестирование представляет собой процесс проверки того, насколько программное обеспечение соответствует требованиям, заявленным заказчиком. Он осуществляется в специальных, искусственно создаваемых ситуациях посредством наблюдения за работой программного обеспечения. Такого рода искусственно построенные ситуации называют тестовыми, или просто тестами.

Разработка тестов происходит на основе проверяемых требований и критерия полноты тестирования. Разработанные тесты формируются в тест-кейс (набор тестов) и выполняются на ПО, которое нужно протестировать. После прогона всех тестов анализируется результат, в результате чего можно выявить ошибки в программе.



*Рисунок 2. Процесс тестирования.*

* + 1. **Разработка тест-кейсов**

**Тест-кейс (тестовый случай)** – это минимальный элемент тестирования (одна проверка), содержащий в себе описание конкретных действий, условий и параметров, которые направленны на проверку какой-либо функциональности. Набор тест-кейсов называется тестовым набором (test suite).

Тест-кейсы позволяют тестировщикам провести проверку продукта без полного ознакомления с документацией. При условии, что созданный тест-кейс удобен в поддержке, то, написанный один раз, он позволит сэкономить большое количество времени и усилий тестировщиков. Подробные тест-кейсы также способны существенно снизить вариативность выполнения тестов различными тестировщиками, что повышает качество тестирования.

**Тест-кейс должен включать в себя:**

* Уникальный идентификатор тест-кейса. Этот идентификатор необходим для удобства организации и навигации по тестовым наборам.
* Название. В названии должна отражаться основная идея тест-кейса, цель данной проверки.
* Предусловия. Список шагов, не имеющих прямого отношения к проверяемому функционалу, которые необходимо выполнить до начала теста. Например, для тест-кейса «Заказ товара» предусловием может быть шаг «авторизоваться на сайте», если на данном сайте заказать товар может только авторизованный пользователь.
* Шаги. Описание последовательности действий, которая должна привести к ожидаемому результату.
* Ожидаемый результат. Поведение системы, предусмотренное требованиями. Один тест-кейс проверяет одну конкретную функцию, поэтому ожидаемый результат может быть только один.
* Статус кейса. Проставляется в соответствии с тем, соответствует ли фактический результат ожидаемому. Тест-кейс может иметь один из трех статусов:
* Положительный, если фактический результат совпадает с ожидаемым результатом.
* Отрицательный, если фактический результат не совпадает с ожидаемым результатом.
* Выполнение теста блокировано, если после одного из шагов продолжение теста невозможно. В этом случае так же отрицательный.
* История редактирования. Дает возможность узнать, кем и когда был изменен тест-кейс. Это удобно, поскольку позволяет более эффективно редактировать тест-кейсы.

**Требования к тест-кейсу**

* Для измерения покрытия функциональных возможностей ПО, требования к продукту должны быть проанализированы и впоследствии разбиты на пункты. Если тест-кейсы покрывают все требования, то может быть дан положительный или отрицательный ответ о реализации данного требования в продукте.
* Тест является корректным в случае, когда он может обеспечить высокую вероятность обнаружения ошибки. Показать, что в программе полностью отсутствуют ошибки невозможно, поэтому процесс тестирования должен быть направлен на выявление прежде не найденных ошибок.
* Четкие, однозначные формулировки шагов. Описание шагов для прохождения тест-кейса должно содержать всю необходимую информацию, но при этом тест-кейс не должен быть слишком детализирован. Например, если тест-кейс содержит такие шаги, как авторизация, в описании необходимо указывать логин и пароль, но не нужно указывать в каком углу экрана находится окно авторизации.
* Отсутствие зависимостей тест-кейсов. Если тесты связанны между собой, становится проблематичным изменение, дополнение или удаление конкретного тест-кейса, появляется необходимость изменять связанные с ним тесты. Более того, взаимосвязанные тесты обладают конкретной последовательностью от перехода одного теста к другому. Это приведет к тому, что не все сценарии перехода от одного теста к другому будут протестированы, и появляется вероятность пропустить баг.
* Ожидаемый результат необходимо прогнозировать заранее и прописывать его в тест-кейсе. Если ожидаемый результат не определить заранее, может возникнуть ситуация, когда тестировщик видит то, что он хочет увидеть. При заранее определенном результате тестировщику необходимо только сравнить ожидаемый результат с фактическим.
* Необходимо уделять внимание не только тестам, которые проверяют правильные данные, но и тем тестам, которые проверяют работу программы при неправильных, непредусмотренных данных. Большое количество ошибок связано именно с теми действиями пользователя, которые не предусмотрены программой.
* Также необходимо проверять, не делает ли программа то, чего не должна. Нужно производить проверку на нежелательные побочные эффекты.

В таблице 1 приведен пример тест-кейса, отвечающего вышеописанным требованиям.

*Таблица 1. Пример тест-кейса.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Тест WB-79: Создание шаблона по p2p переводу** | | | |
| Описание теста: цель, сценарий и исходное состояние программы:  Проверка работы функциональности "Шаблоны" | | | |
| №: | Шаги: | Ожидаемая реакция: | Execution Status: |
| 1 | Авторизоваться на тестовом сервере roox1.open.ru под: **knerdadovsky** | Авторизация прошла успешно. | Пройден |
| 2 | Перейти в пояс “Переводы” | Пояс “Переводы” отображается на экране. Стандартное значение – “Между своими картами и счетами” | Пройден |
| 3 | Нажать на кнопку “По номеру карты” | Отображается форма для осуществления p2p перевода | Пройден |
| 4 | Выбрать рублевую дебетовую карту из выпадающего списка | Карта успешно выбрана и закреплена на форме p2p перевода | Пройден |
| 5 | Ввести номер карты (Например: **5446 3800 7004 5163**) | На форме p2p перевода отобразилось лого банка и платежная система | Пройден |
| 6 | Ввести сумму перевода (Например: **50,07 руб.**) | Автоматически рассчитана комиссия: 0,00 руб. | Пройден |
| 7 | Нажать кнопку “Перевести” | Появляется сообщения о принятие платежа в обработку, а так же кнопка “Создать шаблон” | Пройден |
| 8 | Нажать кнопку “Создать шаблон” | Появилась форма для ввода названия шаблона | Пройден |
| 9 | Ввести название (Например: p2p) | Появляется кнопка “Сохранить” | Пройден |
| 10 | Нажать кнопку “Сохранить” | Появилось сообщение об успешном сохранении шаблона | Пройден |
| Execution type: | Вручную | | |
| Estimated exec. duration (min): | 5.00 | | |
| Приоритет: | Medium | | |
| **Execution Details** |  | | |
| Версия (сборка) | Webapi-2.1 | | |
| Тестировщик | Alekseev\_fm@open.ru | | |
| Execution Result: | **Пройден** | | |
| Execution Mode: | **Вручную** | | |

Данный тест-кейс содержит следующие обязательные атрибуты:

* Уникальный идентификатор тест-кейса: WB-79.
* Название: Создание шаблона по p2p переводу.
* Предусловия: в данном тест-кейсе предусловием является авторизация на портале. Непосредственно к тесту авторизация не относится (авторизацию на портале проверяет отдельный тест-кейс), но без авторизации данная проверка невозможна.
* Шаги.
* Ожидаемый результат.
* Статус кейса.
* История редактирования: в тест-кейсе указана информация о том, кто проходил тест, в какое время и в рамках какой сборки.

Данный тест-кейс отвечает требованиям, изложенным выше. Шаги сформулированы однозначно, содержат необходимую, но не лишнюю информацию, ожидаемый результат четкий и однозначный.

* + 1. **Выполнение тест-кейсов**

Одной из особенностей процесса тестирования является необходимость проведения тестирования программы специалистом, который не является ее автором. Категорически неприемлемо тестирование продукта программистом, создавшим его. Это правило должно быть применимо ко всем, без исключения, формам тестирования, например, как к тестированию целой системы, так и к тестированию отдельных модулей, а также внешних функций. Несомненно, при проведении процесса тестирования не автором программы, он становится более точным и эффективным.

Если рассматривать суть процесса тестирования, то становится очевидным, что это процесс деструктивный. С этой его особенностью и связывают представление о тестировании как о сложной работе. Исключительно трудной эта работа представляется для программиста, создавшего продукт, так как проектирование, разработка и написание программы является процессом конструктивным, в отличие от тестирования, в процессе которого специалисту необходим настрой на деструктивный образ мышления. Исходя из этой особенности тестирования, приступать к тщательному и беспристрастному выявлению ошибок сразу же по завершению создания программы представляется трудновыполнимым для ее автора.

Например, содержащиеся в модуле дефекты, которые являются следствием ошибок перевода (такие как, к примеру, неверная интерпретация спецификации), с высокой долей вероятности будут присутствовать и в тестах, так как оба процесса будут выполняться одним и тем же специалистом. Также могут обнаружиться в тестах и ошибки, сделанные при понимании и сопряжении других модулей.

Однако, не следует делать однозначный вывод, что тестирование программы специалистом, создавшим ее, невозможно. Большое количество программистов справляются с этой задачей вполне успешно. Но, исходя из вышесказанного, можно прийти к заключению, что выполнение тестирования другим специалистом, не являющимся ее автором, гораздо эффективнее и плодотворнее. Поэтому и создаются группы тестировщиков, специально для проведения процесса тестирования с наиболее адекватными результатами.

Принятие решения о сроках остановки тестирования является одной из сложных проблем при проведении тестирования. Это происходит, потому что невозможно провести процесс испытания всех входных значений или, другими словами, исчерпывающее тестирование. Следовательно, при процессе тестирования техническая сторона проводимых действий входит в противоречие с экономической выгодой, так как возникает вопрос выбора оптимального конечного количества тестов, которое дает максимально возможный результат (вероятность обнаружения ошибок) при определенных затратах.

Существует довольно много примеров, когда написанные тесты были малоэффективны, так как их способность обнаружения новых ошибок была крайне низкой. В то же время, хорошие тесты, обнаруживающие ошибки с высокой точностью, могли остаться без внимания специалистов.

Таким образом, принимая решение о количестве времени и проводимых с программой тестов, всегда необходимо принимать во внимание уровень риска, как технического, так и бизнес риска, для отдельно взятого продукта и для всего проекта. Более того, нужно учитывать тот факт, что проект обычно имеет ограничения по времени и бюджету.

* + 1. **Анализ результатов тестирования**

Несмотря на существование различных видов тестирования, процессы тестирования достаточно схожи. Разработкой и анализом тестов может заниматься только тестировщик. За выполнение тест-кейсов также отвечает тестировщик, однако выполнение этих тестов может производиться как вручную, так и в автоматизированном режиме.

По результатам выполнения каждого теста ему присваивается статус (положительный, отрицательный, блокирован). Если тест получает отрицательный статус, то в зависимости от методологии тестирования тестировщик может проводить дополнительную работу для выявления конкретной ошибки, которая была причиной некорректного поведения программы.

При использовании методологии черного ящика, анализ результатов теста сводится к выявлению общих закономерностей, ведущих к появлению ошибки. Однако, когда используется белый или серый ящики, тестировщик может проводить гораздо более глубокий анализ причин возникновения ошибки. В зависимости от доступных тестировщику данных (база данных, исходный код программы, логи и т.п.), он способен с некоторой точностью определить источник некорректного поведения программы.

После того, как максимально точно выявлена причина нежелательного поведения, тестировщик должен описать её вместе со способом воспроизведения ошибки для дальнейшей передачи этой информации разработчикам. Когда источник ошибки точно определен и хорошо описан, разработчикам гораздо проще исправить эту ошибку.

* 1. **Описание процесса автоматизированного тестирования**

Автоматизированное тестирование представляет собой процесс проверки таких функций, как запуск, инициализация, исполнение, анализ и вывод, производимый в автоматическом режиме. Автоматизированное тестирование осуществляется с помощью специального программного обеспечения и вспомогательных инструментов. Этот тип тестирования решает те же задачи, что и ручное функциональное тестирование.

Автоматизация тестирования начинается с построения процесса ручного тестирования, а именно, с документации, написанной для такого тестирования. Таким образом, для того, чтобы начать процесс автоматизации тестирования, нужно точно знать, что и как должно быть реализовано. Обычно каждый автоматизированный тест базируется на тест-кейсе для ручного тестирования с повышенным уровнем детализации.

**Выбор тестов для автоматизации.**

Для того, чтобы определить, что именно нужно автоматизировать, необходимо сначала оценить целесообразность автоматизации тестирования в условиях проекта. Если автоматизация целесообразна, то, принимая во внимание требования к объекту тестирования, необходимо разработать план, согласно которому автоматизированные тесты будут создаваться. При разработке подобного документа, необходимо четко понимать, что нужно автоматизировать и как именно это реализовать. Автоматизация тестирования обычно позволяет ускорить и удешевить процесс тестирования, однако существуют условия, в которых автоматизация особенно эффективна:

* Функциональность, которая используется довольно часто и в которой риски от ошибок достаточно высоки. Автоматизация проверки критической функциональности поможет быстрее находить ошибки, следовательно, быстрее их устранять.
* Рутинные операции, например, проверка форм, в которых необходимо заполнять большое количество полей.
* Труднодоступные места в системе, такие как запись в базу данных, бекэнд процессы, логирование.
* Валидационные сообщения (проверка появления валидации при некорректном заполнении поля).
* Проверка корректного поиска данных.
* Проверки, требующие точных математических расчетов.

И многое другое, в зависимости от инструментов тестирования и требований к системе.

Можно выделить два основных вида автоматизированного тестирования: тестирование API и GUI тесты.

* **Тестирование API** осуществляет проверку взаимодействия нескольких модулей между собой, или проверяет взаимодействие со сторонними сервисами.
* **Тестирование GUI** – тестирование через графический пользовательский интерфейс. С помощью имитации действий пользователя проверяются такие аспекты, как надлежащее функционирование кнопок и полей, правильное появление диагностических сообщений об ошибках и корректная работа приложения в целом.

Сегодня существует множество средств, позволяющих автоматизировать тестирование программного продукта. Однако автоматизация тестирования не всегда востребована, ведь в конечном итоге может оказаться, что разработка и сопровождение систем автоматизированного тестирования обойдется компании дороже, чем труд ручных тестировщиков.

При анализе целесообразности создания систем автоматизированного тестирования важно оценить:

* насколько сложно эту работу выполнять “руками”;
* как часто будут выполняться эти тесты;
* необходимо ли увеличивать скорость выполнения тестов;
* если тесты нужно автоматизировать, то нужно ли это делать со всеми тестами, или достаточно автоматизировать лишь некоторые из них.

Основные задачи, которые решаются внедрением автоматизированного тестирования – это снижение времени, нужного для тестирования и повышение качества тестирования.

Под повышением качества подразумевается, что автоматизированное тестирование обычно позволяет покрыть большее количество функционала, а также позволяет выполнять тесты, которые невозможно осуществлять вручную. Автотесты также дают возможность минимизировать присутствие человеческого фактора при тестировании.

Следовательно, можно выделить основные плюсы и минусы автоматизированного тестирования.

Преимущества:

* Сокращение времени, затрачиваемого на исполнение тестов (относительно ручного тестирования).
* Возможность проводить тесты, которые невозможно реализовать без использования программных средств автоматизации.
* Экспертиза становится более объективной, поскольку исключается человеческий фактор.

Недостатки:

* Автоматизация тестирования может потребовать очень значительных трудозатрат.
* Требуется персонал с гораздо более высокой квалификацией.
* Более сложный анализ результатов, из-за необходимости понимания ошибок, выводимых при падении теста.
* Повышение трудозатрат на актуализацию автотестов при изменениях в системе.
* Риск появления ошибок в самом автотесте.
* Не все тест-кейсы возможно автоматизировать. Например, проверка правильного отображения информации в выписке банковского счета в формате PDF.
  1. **Критерии эффективности процесса автоматизированного тестирования**

Процесс тестирования должен быть эффективен в первую очередь с точки зрения компании, в которой он протекает. Компании могут быть интересны следующие параметры процесса тестирования:

* Время, необходимое для разработки тестов.
* Время, которое занимает один цикл тестирования.
* Квалификация персонала, необходимая для разработки и проведения тестов.

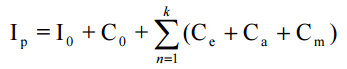
Изменив любой из этих параметров, компания может повлиять на качество тестирования. Однако важно понимать, что любая комбинация этих параметров может быть выражена в денежном эквиваленте, и, как правило, у любого конкретного процесса тестирования есть оптимальная комбинация, при которой достигается достаточный уровень качества тестирования при минимальных денежных затратах.

Автоматизируя процесс тестирования, мы, разумеется, меняем процесс тестирования, а вместе с ним поменяется и оптимальная комбинация перечисленных выше параметров. Например, можно ожидать, что увеличится время, необходимое для разработки тестов и повысятся требования к квалификации персонала, при этом сильно понизится время, занимаемое одним циклом тестирования. Учитывая, что комбинация параметров стала новой, вероятно, поменяется и качество тестирования вместе с его стоимостью.

Для того, чтобы была возможность дать численный эквивалент эффективности процесса тестирования, предлагается зафиксировать параметр качества на определенном уровне. Тогда численной оценкой эффективности определенного способа тестирования будет являться величина инвестиций, необходимая для того, чтобы он обеспечивал некий определенный уровень качества.

Оценка целесообразности автоматизации тестирования производится с помощью подсчета затрат на ручное и автоматизированное тестирование и их сравнение. Точно посчитать финансовую целесообразность автоматизации тестов обычно невозможно, поскольку она зависит от параметров, которые в процессе разработки продукта могут быть лишь примерно понятно (например, планируемая длина жизненного цикла системы или точный список тестов, подлежащих автоматизации).

Для расчёта инвестиций, необходимых для внедрения и эксплуатации автоматизированных тестов за выделенный период (**Ip**), используется следующая формула:



**I0** - Оценка стартовых инвестиции, которые состоят из затрат на лицензии необходимого программного обеспечения для разработки автотестов, стоимости дополнительных аппаратных средств и т.п.

**C0** - Оценка стоимости разработки и отладки библиотеки автоматических тестов, которая рассчитывается как произведение среднего времени, нужного для написания одного автоматизированного теста одним разработчиком тестов (в часах), умноженное на цену его рабочего часа и на общее количество тестов, которые предстоит автоматизировать.

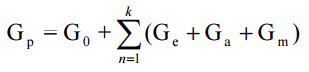
**k** - Это количество планируемых прогонов тестов (циклов тестирования) за всё оставшееся время жизненного цикла продукта.

**Ce** - Оценка стоимости одного прогона всех автоматизированных тестов, которая рассчитывается как время, необходимое для подготовки к выполнению тестирования, сложенное со средним временем выполнения одного теста одним тестировщиком, умноженное на цену рабочего часа и на общее количество тестов. В нашем случае эта переменная принята за 0, поскольку подготовка к циклу тестирования не требуется, а само тестирование не требует дополнительного контроля со стороны работника и происходит полностью автономно.

**Ca** - Оценка затрат на анализ результатов одной итерации цикла автоматизированного тестирования, которая вычисляется как оценка доли отрицательных тестов, умноженная на количество тестов, на среднее время, необходимое для анализа причин отрицательной оценки одного теста одним тестировщиком, и на цену одного рабочего часа тестировщика.

**Cm** - Оценка стоимости поддержания автоматизированных тестов в рабочем и актуальном состоянии. Рассчитывается как вероятность появления необходимости изменения одного теста между циклами тестирования, умноженная на количество тестов, на среднее время, необходимое для актуализации одного теста и на цену одного рабочего часа тестировщика.

Оценка стоимости ручного тестирования (**Gp**) представлена в следующей формуле:



**G0** - Оценка стоимости разработки базы тест-кейсов для ручного тестирования.

**k** - Это количество планируемых прогонов тестов (циклов тестирования) за всё оставшееся время жизненного цикла продукта.

**Ge** - Оценка стоимости однократного выполнения цикла ручного тестирования, которая рассчитывается как среднее время, затрачиваемое на подготовку к тестированию плюс среднее время, нужное для выполнения одного тест-кейса одним тестировщиком, умноженное на суммарное количество кейсов и на цену одного рабочего часа тестировщика.

**Ga** - Оценка стоимости анализа результатов для одного прогона цикла ручного тестирования. Вычисляется как оценка средней доли отрицательных тестов в прогоне, умноженная на количество тестов, на среднее время, необходимое для анализа причин отрицательной оценки одного теста одним тестировщиком, и на цену одного рабочего часа тестировщика;

**Gm** - Оценка стоимости поддержания ручных тестов в актуальном состоянии. Рассчитывается как вероятность появления необходимости изменения одного теста между циклами тестирования, умноженная на количество тестов, на среднее время, необходимое для актуализации одного теста и на цену одного рабочего часа тестировщика.

С помощью этих формул компания может посчитать, будет ли внедрение автоматизированных тестов экономически оправдано. В следующей главе на конкретном примере будет рассчитана экономическая целесообразность внедрения таких тестов.

**Глава 2. Автоматизация процесса тестирования**

В этой главе на конкретном примере будет проверяться целесообразность автоматизации тестирования в компании. Также будет создан фреймворк с реализованными автоматизированными тестами двух типов: API – тесты, GUI – тесты.

* 1. **Описание компании**

Для данного исследования мною была выбрана компания ПАО Банк «ФК Открытие». Банк «ФК Открытие» входит в топ-10 крупнейших банков России и является системно-значимым банком. Банк работает на финансовом рынке с 1993 года. «ФК Открытие» — универсальный банк с устойчивой диверсифицированной структурой бизнеса. Банк развивает следующие ключевые направления бизнеса: корпоративный, инвестиционный, розничный, МСБ и Private Banking. В компании «ФК Открытие» существует большой отдел тестирования и контроля качества, состоящий из 76 специалистов. В отделе есть специалисты, занимающиеся статическим тестированием, тестированием документации, также специалисты, занимающийся динамическим тестированием интернет банка. Каждые две недели в продуктивную среду выпускаются исправления и улучшения интернет банка. Перед выпуском релиза в продуктив, на тестовой среде проводится регрессионное тестирование, для того, чтобы убедиться, что основной функционал сайта не поврежден и в продуктив выйдет корректный релиз.

На данный момент регрессионное тестирование состоит из 120 проверок, и ни один тест не автоматизирован.

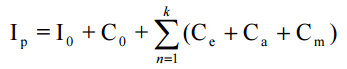
* 1. **Расчёт экономической целесообразности введения автоматизированного тестирования**

Для проверки целесообразности автоматизации процесса тестирования в компании необходимо посчитать затраты на ручное тестирование и затраты на автоматизацию. Расчеты будут производиться исходя из данных, полученных в ходе опроса работников отдела тестирования и контроля качества в компании «ФК Открытие». При опросе было выявлено, что регрессионное тестирование, которое проводится каждые две недели, занимает несколько дней у тестировщиков и именно этот вид тестирования специалисты данного отдела хотели бы автоматизировать. Для расчета целесообразности автоматизации используются формулы, описанные во второй главе. В ходе опроса были получены данные, необходимые для подсчетов.

* Оплата тестировщика, занимающегося автоматизацией, оценивается в 600 рублей в час, в то время как оплата ручного тестировщика составляет 500 рублей в час.
* Данный проект рассчитан как минимум еще на три года. Регрессионное тестирование проводится каждые две недели, но часто случается так, что после исправления критичных ошибок, найденных при тестировании, проверки необходимо выполнять заново. И того, примерно 1.5 проведения регрессионного тестирования в две недели. При тестировании используется 120 тестов.
* На подготовку к регрессионному тестированию у ведущего тестировщика обычно уходит порядка 2 часов, преимущественно это время тратится на распределение задач между тестировщиками и другие организационные задачи. Среднее время, необходимое одному тестировщику на выполнение одного тест-кейса, составляет 10 минут.
* При каждом прогоне примерно 5% тестов имеют отрицательные результаты. На определение источника ошибки для каждого теста у ручного тестировщика уходит около 10 минут, в то время как при автоматизированном тестировании анализ ошибки занимает 15 минут. При ручном тестировании тестировщик сразу видит, где именно и при каких входных данных произошла ошибка, а при автоматизированном тестировании эту информацию необходимо искать в коде.
* Вероятность появления необходимости изменения одного теста между циклами тестирования оценена в 3%, Среднее время, необходимое для актуализации одного теста около 6 минут. Для актуализации автоматизированного теста потребуется 30 минут.
* Автоматизация одного теста оценивается в 3 часа

Учитывая информацию, полученную в ходе опроса специалистов отдела тестирования, можно произвести расчет затрат на ручное и автоматизированное тестирование.

Формула для расчета затрат на автоматизированное тестирование



Начальные инвестиции в данном случае равны нулю, поскольку используется бесплатный набор технологий (Интегрированая среда разработки, Фреймфорки и прочее) и отсутствует необходимость вкладываться в дополнительное оборудование.

Стоимость разработки автоматизированных тестов равна 216 000 рублей (120 тестов \* 3 часа \* 600 руб/час).

Планируемое количество циклов тестирования - 126 раз (3 года \* 28 недель \* 1.5 раза)

Оценка стоимости однократного выполнения цикла автоматизированного тестирования равна нулю, поскольку подготовка к циклу тестирования не требуется, а само тестирование не нуждается в дополнительном контроле со стороны работника и происходит полностью автономно.

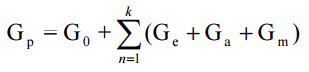
Оценка стоимости анализа результатов выполненного цикла автоматизированного тестирования равна 900 рублей (120 тестов \* 0.05 \* 0.25 часа \* 600 руб/час)

Оценка стоимости поддержания автоматизированных тестов в рабочем и актуальном состоянии равна 1 080 рублей (120 тестов \* 0.03 \* 0.5 часа \* 600 руб/час).

Таким образом, итоговая стоимость внедрения и эксплуатации системы автоматизированных тестов равна:

0 + 216 000 + 126 \* (0 + 900 + 1 080) = 465 900 рублей.

Формула для расчета затрат на ручное тестирование:



Оценка стоимости разработки базы тест-кейсов для ручного тестирования равна нулю, поскольку компания уже обладает базой тест-кейсов

Оценка стоимости однократного выполнения цикла ручного тестирования равна 10 575 рублей (0.75 + 120 тестов\* 0.17) \* 500руб/час.

Оценка стоимости анализа результатов для одного прогона цикла ручного тестирования равна 510 рублей (120 \* 0.05 \* 0.17 \* 500).

Оценка стоимости поддержания ручных тестов в актуальном состоянии равна 180 рублей (120 \* 0.03 \* 0.1 \* 500) .

Итоговая стоимость затрат на ручное тестирование равна:

0 + 126 \* (10 575 + 510 + 180) = 1 419 390 рублей.

Таким образом затраты на автоматизированное тестирования будут в 3 раза меньше, чем на ручное. Следовательно, можно прийти к заключению, что на данном проекте внедрение автоматизации тестирования целесообразно. Далее будет разработан фреймворк с реализованными автоматизированными тестами двух типов: API – тесты, GUI – тесты..

* 1. **Внедрение автоматизированных тестов**

Автоматизация функциональных тестов веб-приложений обычно происходит с помощью специальных библиотек, позволяющих симулировать поведение реальных пользователей. Эти библиотеки обычно состоят из 2-х частей: программы или надстройки над браузером, которая позволяет управлять браузером и выполнять команды внутри него и программного API, предоставляющего удобные функции для контролирования этой программы. Ниже описаны основные функции, которые предоставляют библиотеки, которые используются для автоматизации тестирования:

* Общие функции браузера (Открытие новой вкладки или окна и контролирование их размеров).
* Навигация между веб страницами.
* Поиск веб элемента на странице с указанными параметрами.
* Функции ожидания определенных событий (например ожидание полной загрузки страницы или появления на ней определенного элемента).
* Симуляция действий пользователя, таких как нажатие кнопки мыши на определенный элемент или ввод последовательности символов с клавиатуры.
* Для более сложных действий предоставляется возможность исполнять JavaScript команды на странице.
* Выполнение HTTP запросов
* Десериализация HTTP ответа в Java класс

Автоматизировать тесты можно с помощью различных библиотек. Есть большое количество инструментов для автоматизации функционального тестирования, основные из которых: Selenium WebDriver, [Watij](http://watij.com/), [HtmlUnit](http://htmlunit.sourceforge.net/) [Jamaleon](http://jameleon.sourceforge.net/), Selenide, RestAssured.

Мною были выбраны библиотеки: Selenide – для **GUI** тестирования; RestAssured – для **API** тестирования. Эти библиотеки имеют большую популярность среди специалистов, работающих в области автоматизации тестирования, и обладают достаточным количеством возможностей для написания фреймворка с реализованными функциональными тестами. Из возможных языков программирования для реализации тестов (Java, C#, Ruby, Python) был выбран Java, поскольку на момент написания данной работы мною был накоплен больший опыт использования этого языка, чем других. Также при реализации фреймворка будет использоваться библиотека AllureReport, разработанная компанией Яндекс. Эта библиотека генерирует отчет о прохождении автотестов и позволяет значительно уменьшить время на анализ результатов.

При практическом применении автоматизированных функциональных тестов очень быстро обнаруживается, что тестирование каждого отдельного веб приложения обладает своей спецификой. Это подталкивает к созданию дополнительного программного слоя между библиотеками для тестирования и автотестами, или же фреймворка. Такой подход позволяет минимизировать количество повторяющегося кода путем вынесения его в отдельные пакеты и классы фреймворка, что положительно сказывается на скорости создания автотестов, понижает вероятность возникновения ошибок и улучшает удобочитаемость кода. Часто в автотестах может понадобиться протестировать функционал, для которого необходимо авторизоваться, найти типичный для всех страниц приложения элемент, сделать HTTP запрос к API веб приложения, чтобы сверить данные с представленными на странице интернет банка и т.д. Такие действия будут регулярно использоваться во многих автотестах, при этом, если не вынести их в отдельные классы и методы, которые в дальнейшем будут переиспользоваться, они будут только загромождать код.

Ниже представлены реальные тест-кейсы для тестирования сайта интернет банка в компании «ФК Открытие», описанные в программе Microsoft Team Foundation Server 2017. Так как тестовые сервера доступны только из внутренней сети банка, в тест-кейсах будет использоваться продуктивный сервер интернет банка.

Данный тест проверяет работу сервиса «Переводы» с помощью API . Для проверки данного функционала необходим пользователь с картой и накопительным счетом. Необходимо авторизоваться под пользователем. Отправить запросы на получение карт и счетов. Выбрать необходимые продукты и выполнить запрос на перевод. Проверить ответ от сервера на обязательные поля и соответствие значения этих полей ожидаемым.

*Таблица 2. Тест-кейс «Переводы»*

|  |  |  |  |
| --- | --- | --- | --- |
| **Тест WB-29: API Переводы: перевод между своими продуктами (Моя копилка RUR – 3C RUR)** | | | |
| Описание теста: цель, сценарий и исходное состояние программы:  Проверка работы функциональности "Переводы" | | | |
| №: | Шаги: | Ожидаемая реакция: | Execution Status: |
| 1 | Авторизоваться на сервере ib.open.ru под: **alekseevfm** | Авторизация прошла успешно. AccessToken получен. | Пройден |
| 2 | Послать запрос на получения списка карт. | Статус ответа: 200. Поле success = true. Ответ содержит в себе хотя бы одну карту | Пройден |
| 3 | Выбрать карту с параметрами: bsc = 3C, currency = RUR, code = NORMAL и балансом больше 100 рублей | Необходимая карта найдена | Пройден |
| 4 | Послать запрос на получения списка накопительных счетов. | Статус ответа: 200. Поле success = true. Ответ содержит в себе хотя бы один накопительный счет | Пройден |
| 5 | Выбрать счет с параметрами: AccName = Моя копилка, currency = RUR, code = NORMAL и балансом больше 100 рублей | Необходимый счет найден. | Пройден |
| 6 | Послать запрос на перевод между выбранными продуктами на сумму 1 руб. | Статус ответа: 200. Поле success = true | Пройден |
| 7 | Проверить поля в ответе: recieptable = true, templatable = true | Проверка успешна | Пройден |
| Execution type: | Вручную | | |
| Estimated exec. duration (min): | 5.00 | | |
| Приоритет: | Medium | | |
| **Execution Details** |  | | |
| Версия (сборка) | Webapi-2.1 | | |
| Тестировщик | Alekseev\_fm@open.ru | | |
| Execution Result: | **Пройден** | | |
| Execution Mode: | **Вручную** | | |

Этот тест проверяет работоспособность шаблонов в интернет банке по средствам UI. Для проверки данного функционала необходим пользователь с шаблоном на переводы между своими продуктами. Необходимо авторизоваться под пользователем. Открыть список всех шаблонов и выбрать шаблон на перевод между своими продуктами. Далее необходимо подтвердить перевод и проверить успешность перевода.

*Таблица 3. Тест-кейс «Шаблоны»*

|  |  |  |  |
| --- | --- | --- | --- |
| **Тест WB-47: UI Шаблоны: Применение шаблона по переводу между своими продуктами** | | | |
| Описание теста: цель, сценарий и исходное состояние программы:  Проверка работы функциональности "Переводы" | | | |
| №: | Шаги: | Ожидаемая реакция: | Execution Status: |
| 1 | Авторизоваться на сервере ib.open.ru под: **alekseevfm** | Авторизация прошла успешно. | Пройден |
| 2 | В шапке нажать на <...> | Произошел переход на страницу со всеми шаблонами | Пройден |
| 3 | Выбрать шаблон с названием «Между своими» | Произошёл переход в пояс "Переводы", заполнены поля для выполнения платежа. | Пройден |
| 4 | Нажать "Перевести" | Произошел переход на форму подтверждения | Пройден |
| 5 | Нажать "Подтвердить" | Появилась иконка загрузки, затем окно успешного завершения перевода. | Пройден |
| Execution type: | Вручную | | |
| Estimated exec. duration (min): | 5.00 | | |
| Приоритет: | Medium | | |
| **Execution Details** |  | | |
| Версия (сборка) | Webapi-2.1 | | |
| Тестировщик | [Alekseev\_fm@open.ru](mailto:Alekseev_fm@open.ru) | | |
| Execution Result: | **Пройден** | | |
| Execution Mode: | **Вручную** | | |

Далее представлена реализация описанных выше тестов.

**API Переводы: перевод между своими продуктами (Моя копилка RUR – 3C RUR)**

**Класс для тестов на переводы и тестовый метод**

@Features(**"Переводы"**)*//аннотация для отчета***public class** TransferTests **extends** Authorization {  
  
 @Stories(**"Critical тесты"**)*//аннотация для отчета* @Test  
 @Title(**"Перевод между своими продуктами (Моя копилка RUR - ГО RUR )"**)*//аннотация для отчета* **public void** transferBetweenMy() {  
 BigDecimal amount = **new** BigDecimal(1);  
 TestUser testUser = **new** TestUser(**"основной"**);  
 login(testUser);  
 CardDTO cards = **GET\_REQUEST**.getRequest(CardPaths.***CARD***).as(CardDTO.**class**);  
 CardData cardData = **CARD\_STEPS**.findCard(cards, **new** Properties());  
 AccumulationDTO accumulationDTO = **GET\_REQUEST**.getRequest(AccountPaths.***ACCUMULATION\_LIST***).as(AccumulationDTO.**class**);  
 AccumulationData accumulationData = **ACCUMULATION\_STEPS**.findAccumulation(accumulationDTO,  
 **new** Properties().setAmount(**new** BigDecimal(100)).setAccName(**"Моя копилка"**));  
 TransferInfoDTO transferInfo = **POST\_REQUEST**.postRequest(InternalPaths.***ACCOUNT\_TO\_CARD***,  
 **new** AccountToCard(amount, cardData.getBalance().getCurrency(), cardData.getVirtualCardNum(),  
 amount, accumulationData.getBalance().getCurrency(), accumulationData.getAccNum())).as(TransferInfoDTO.**class**);  
 **INTERNAL\_STEPS**.checkTransferInfo(transferInfo);  
 }  
}

**Метод для выбора пользователя**

@Step(**"Выбрать пользователя c ролью: {0}"**)  
**public void** readFileSuperUsers(String role) {  
 String line;  
 String[] sSpl;  
 ArrayList<String> loginArray = **new** ArrayList<String>();  
 ArrayList<String> passwordArray = **new** ArrayList<String>();  
 ArrayList<String> fioArray = **new** ArrayList<String>();  
 Map<String, String> userInfo = **new** HashMap<String, String>();  
 **try** {  
 String curDir = **new** File(**""**).getAbsolutePath();  
 BufferedReader br = **null**;  
 **try** {  
 br = **new** BufferedReader(**new** FileReader(curDir + **"\\src\\main\\resources\\testUsers.csv"**));  
 **while** ((line = br.readLine()) != **null**) {  
 **if** (line.contains(role)) {  
 sSpl = line.split(**";"**);  
 loginArray.add(sSpl[2]);  
 passwordArray.add(sSpl[3]);  
 fioArray.add(sSpl[4]);  
 }  
 }  
 } **finally** {  
 br.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 Assert.*assertNotEquals*(loginArray.size(), 0, **"Пользователь с необходимой ролью не найден"**);  
 **this**.**username** = loginArray.get(0);  
 userInfo.put(**"username"**, **username**);  
 **this**.**password** = passwordArray.get(0);  
 userInfo.put(**"password"**, **password**);  
 **this**.**fio** = fioArray.get(0);  
 userInfo.put(**"fio"**, **fio**);  
 Utils.*saveUserInfo*(**"Информация о полученном пользователе"**, userInfo);  
}

**Метод для авторизации**

@Step(**"Авторизация пользователем: {0}"**)  
**public void** login(String user, String password) {  
 Response response = **POST\_REQUEST**.postRequestForAuthorization(GeneralPaths.***ACCESS\_TOKEN***, **new** AccessToken().toString());  
 Map<String, String> cookies = **new** HashMap<>(response.cookies());  
 response = **POST\_REQUEST**.postRequestForAuthorization(GeneralPaths.***ACCESS\_TOKEN***, **new** AccessTokenNext  
 (response.path(**"execution"**), password, user).toString(), cookies);  
 Assert.*assertNotNull*(response.path(**"access\_token"**), **"Не удалось авторизоваться!"**);  
 **logger**.info(**"accessToken ПОЛУЧЕН"**);  
 RequestUtils.*cookieMap*.put(**"at"**, response.path(**"access\_token"**));  
}

**Метод для выполнения GET запросов**

@Step(**"Посылаем GET запрос на сервер: {0}"**)  
**public** Response getRequest(String path) {  
 Response response = *given*().config(**cfg**).accept(**"\*/\*"**).cookies(*cookieMap*).get(generateURI(path));  
 **logger**.info(**"URL: "** + generateURI(path).toString());  
 **logger**.info(**"HTTP STATUS: "** + response.getStatusCode());  
 **logger**.info(**"RESPONSE: "** + response.asString());  
 Assert.*assertEquals*(response.getStatusCode(), 200, **"Статус отввета отличается от \"200\""**);  
 attachJson(**"Тело ответа"**, gsonFormatter(response));  
 **if** (response.path(**"success"**).equals(**true**)) {  
 **return** response;  
 } **else if** (response.path(**"success"**).equals(**false**) && response.path(**"code"**).toString().equals(**"EXCEPTION"**)) {  
 **throw new** IllegalStateException(**"Запрос по URL: "** + generateURI(path) + **" возвратил EXCEPTION"**);  
 } **else** {  
 **throw new** IllegalStateException(**"Ответ от сервера не пришел: "** + generateURI(path));  
 }  
}

**Метод для поиска карты с параметрами**

@Step(**"Поиск карты по параметрам"**)  
**public** CardData findCard(CardDTO card, Properties properties) {  
 **boolean** founded = **true**;  
 CardData foundedCard = **new** CardData();  
 **for** (CardData cardData : card.getData()) {  
 **if** (properties.getAmount() != **null** && properties.getAmount().compareTo(cardData.getBalance().getAmount()) != -1) founded = **false**;  
 **if** (properties.getCurrency() != **null** && !properties.getCurrency().equals(cardData.getBalance().getCurrency())) founded = **false**;  
 **if** (properties.getBsc() != **null** && !properties.getBsc().equals(cardData.getBsc())) founded = **false**;  
 **if** (!properties.getCode().equals(cardData.getStatus().getCode())) founded = **false**;  
 **if** (founded) {  
 foundedCard = cardData;  
 **break**;  
 }  
 }  
 **return** foundedCard;  
}

**Метод для поиска счета с параметрами**

@Step(**"Поиск накопительного счета по параметрам"**)  
**public** AccumulationData findAccumulation(AccumulationDTO accumulation, Properties properties) {  
 **boolean** founded = **true**;  
 AccumulationData foundedAccumulation = **new** AccumulationData();  
 **for** (AccumulationData data : accumulation.getData()) {  
 **if** (properties.getAmount() != **null** && properties.getAmount().compareTo(data.getBalance().getAmount()) != -1) founded = **false**;  
 **if** (properties.getCurrency() != **null** && !properties.getCurrency().equals(data.getBalance().getCurrency())) founded = **false**;  
 **if** (properties.getBsc() != **null** && !properties.getBsc().equals(data.getBsc())) founded = **false**;  
 **if** (properties.getAccName() != **null** && !properties.getAccName().equals(data.getAccName())) founded = **false**;  
 **if** (!properties.getCode().equals(data.getStatus().getCode())) founded = **false**;  
 **if** (founded) {  
 foundedAccumulation = data;  
 **break**;  
 }  
 }  
 **return** foundedAccumulation;  
}

**Метод для выполнения POST запросов**

@Step(**"Посылаем POST запрос на сервер: {0}"**)  
**public** Response postRequest(String path, Object body) {  
 Response response = *given*().config(**cfg**).cookies(*cookieMap*).contentType(**"application/json"**).with().body(**gson**.toJson(body)).post(generateURI(path));  
 **logger**.info(**"URL: "** + generateURI(path).toString());  
 **logger**.info(**"HTTP STATUS: "** + response.getStatusCode());  
 **logger**.info(**"RESPONSE: "** + response.asString());  
 Assert.*assertEquals*(response.getStatusCode(), 200, **"Статус отввета отличается от \"200\""**);  
 attachJson(**"Тело запроса"**, **gson**.toJson(body));  
 attachJson(**"Тело ответа"**, gsonFormatter(response));  
 **if** (response.path(**"success"**).equals(**true**)) {  
 **return** response;  
 } **else if** (response.path(**"success"**).equals(**false**) && response.path(**"code"**).toString().equals(**"EXCEPTION"**)) {  
 **throw new** IllegalStateException(**"Запрос по URL: "** + generateURI(path) + **" возвратил EXCEPTION"**);  
 } **else** {  
 **throw new** IllegalStateException(**"Ответ от сервера не пришел: "** + generateURI(path));  
 }  
}

**Метод для проверки успешности перевода**

@Step(**"Проверка TransferInfoDTO на обязательные поля и возможность повторения"**)  
**public void** checkTransferInfo(TransferInfoDTO transferInfo) {  
 Assert.*assertNotNull*(transferInfo.getData().getIdDigital(), **"Отсутствует idDigital"**);  
 Assert.*assertEquals*(transferInfo.getData().getReceiptable(), **true**,   
 **"Отсутсвует возможность повторения операции"**);  
 Assert.*assertEquals*(transferInfo.getData().getTemplatable(), **true**,   
 **"Отсутсвует возможность создания шаблона операции"**);  
 **logger**.info(**"Проверка TransferInfoDTO прошла успешно"**);  
}

**UI Шаблоны: Применение шаблона по переводу между своими продуктами**

**Класс для тестирования шаблонов и тестовый метод**

@Features(**"Шаблоны"**)  
**public class** TemplatesTests **extends** Base {  
  
 @Stories(**"Critical тесты"**)  
 @Test  
 @Title(**"Применение шаблона по переводу между своими продуктами"**)  
 **public void** transfer() {  
 TestUser testUser = **new** TestUser(**"основной"**);  
 **LOGIN\_PAGE**.login(testUser);  
 **TEMPLATE\_WIDGET**.checkTemplateWidget();  
 **TEMPLATE\_WIDGET**.clickDivMore();  
 **TEMPLATE\_WIDGET**.findTemplate(**"Между своими"**);  
 **TRANSFER\_WIDGET**.clickBtnTransfer();  
 **TRANSFER\_WIDGET**.checkTransferSuccess();  
 }  
}

**Метод для выбора пользователя**

@Step(**"Выбрать пользователя c ролью: {0}"**)  
**public void** readFileSuperUsers(String role) {  
 String line;  
 String[] sSpl;  
 ArrayList<String> loginArray = **new** ArrayList<String>();  
 ArrayList<String> passwordArray = **new** ArrayList<String>();  
 ArrayList<String> fioArray = **new** ArrayList<String>();  
 Map<String, String> userInfo = **new** HashMap<String, String>();  
 **try** {  
 String curDir = **new** File(**""**).getAbsolutePath();  
 BufferedReader br = **null**;  
 **try** {  
 br = **new** BufferedReader(**new** FileReader(curDir + **"\\src\\main\\resources\\testUsers.csv"**));  
 **while** ((line = br.readLine()) != **null**) {  
 **if** (line.contains(role)) {  
 sSpl = line.split(**";"**);  
 loginArray.add(sSpl[2]);  
 passwordArray.add(sSpl[3]);  
 fioArray.add(sSpl[4]);  
 }  
 }  
 } **finally** {  
 br.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 Assert.*assertNotEquals*(loginArray.size(), 0, **"Пользователь с необходимой ролью не найден"**);  
 **this**.**username** = loginArray.get(0);  
 userInfo.put(**"username"**, **username**);  
 **this**.**password** = passwordArray.get(0);  
 userInfo.put(**"password"**, **password**);  
 **this**.**fio** = fioArray.get(0);  
 userInfo.put(**"fio"**, **fio**);  
 Utils.*saveUserInfo*(**"Информация о полученном пользователе"**, userInfo);  
}

**Методы и элементы для авторизации**

**private** SelenideElement **fldUsername** = *$*(By.*xpath*(**"//\*[@id=\"username\"]"**));  
**private** SelenideElement **fldPassword** = *$*(By.*xpath*(**"//\*[@id=\"password\"]"**));  
**private** SelenideElement **btnLogin** = *$*(By.*xpath*(**"//\*[@id=\"login-standart\"]/form/div[5]/div[1]/button"**));  
**private** SelenideElement **divErrorMessage** = *$*(By.*xpath*(**"//\*[@id=\"login-standart\"]/form/div[6]"**));

@Step(**"Логинимься пользователем: {0}"**)  
**private void** login(String username, String password) {  
 openHomePage(Base.*url*);  
 inputToFldUsername(username);  
 inputToFldPassword(password);  
 pressBtnLogin();  
}

@Step(**"Открыть браузер на странице: {0}"**)  
**public void** openHomePage(String url) {  
 Selenide.*open*(url);  
}  
  
@Step(**"Ввести \"{0}\" в поле \"Логин\""**)  
**public void** inputToFldUsername(String username) {  
 **try** {  
 **fldUsername**.setValue(username);  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось ввести значение в поле \"Логин\": "** + e);  
 }  
}  
  
@Step(**"Ввести значение в поле \"Пароль\""**)  
**public void** inputToFldPassword(String password) {  
 **try** {  
 **fldPassword**.setValue(password);  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось ввести значение в поле \"Пароль\": "** + e);  
 }  
}  
  
@Step(**"Нажать кнопку \"Войти\""**)  
**public void** pressBtnLogin() {  
 **try** {  
 **btnLogin**.click();  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось произвести нажатие на кнопку \"Войти\": "** + e);  
 }  
}

**Метод и элементы для проверки видимости шаблонов**

**private** SelenideElement **divTemplateWidget** = *$*(By.*xpath*(**"//div[contains(@class, 'wocb-wrapper-section wocb-grid wocb-grid-middle wocb-grid-wide')]/div[contains(@class, 'wocb-header\_\_templates')]"**));

@Step(**"Проверка виджета шаблонов на видимость"**)  
**public void** checkTemplateWidget() {  
 **try** {  
 **divTemplateWidget**.shouldBe(Condition.***visible***);  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Проверка на видимость прошла не успешно: "** + e);  
 }  
}

**Метод и элементы для нажатия кнопки открытия всех шаблонов**

**private** SelenideElement **divMore** = *$*(By.*xpath*(**"//div[contains(@class, 'wocb-more-link')]"**));

@Step(**"Нажатие на кнопку \"...\""**)  
**public void** clickDivMore(){  
 **try** {  
 **divMore**.click();  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось нажать на кнопку \"...\": "** + e);  
 }  
}

**Метод и элементы для поиска нужного шаблона**

**private** ElementsCollection **divTemplateList** = *$$*(By.*xpath*(**"//div[contains(@class, 'wocb-shortcut wocb-shortcut\_template')]/div[contains(@class, 'wocb-shortcut\_\_container')]"**));

@Step(**"Выбрать шаблон с названием {0}"**)  
**public void** findTemplate(String name){  
 **try** {  
 **for**(SelenideElement element : **divTemplateList**){  
 **if**(element.getText().contains(name)){  
 element.click();  
 **break**;  
 }  
 }  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось выбрать шаблон с нужным названием: "** + e);  
 }  
}

**Метод и элементы для подтверждения перевода**

**private** SelenideElement **btnTransfer** = *$*(By.*xpath*(**"//div[contains(@class, 'wocb-button wocb-button\_confirmable')]"**));

@Step(**"Нажать на кнопку \"Перевести\" и кноку \"Подтвердить\""**)  
**public void** clickBtnTransfer() {  
 **try** {  
 **divInput**.shouldBe(Condition.*not*(Condition.***empty***));  
 **btnTransfer**.click();  
 **btnTransfer**.click();  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось нажать на кнопку \"Перевести\" и кноку \"Подтвердить\": "** + e);  
 }  
}

**Метод и элементы для проверки успешности перевода**

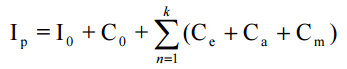
**private** SelenideElement **divMessageSuccess** = *$*(By.*xpath*(**"//div[contains(@class, 'wocb-message wocb-message\_success')]"**));

@Step(**"Проверить сообщение \"Перевод выполнен успешно\""**)  
**public void** checkTransferSuccess() {  
 **try** {  
 **boolean** check = **divMessageSuccess**.shouldBe(Condition.***visible***).getText().contains(**"Перевод выполнен успешно"**);  
 Assert.*assertTrue*(check, **"Сообщение не соответсвует условию"**);  
 } **catch** (Throwable e) {  
 **throw new** IllegalStateException(**"Не удалось проверить сообщение \"Перевод выполнен успешно\": "** + e);  
 }  
}

После того как были автоматизированы несколько тестов, описанных выше, можно сделать вывод о целесообразности автоматизации в контексте компании «ФК Открытие».

Разработка автоматизированных тестов состояла из двух этапов: разработка промежуточного слоя между библиотеками для тестирования и самими автотестами, и реализация тест-кейсов. На первый этап ушло около 16 часов рабочего времени, а на второй около 8. При этом было разработано 2 теста.

При расчете затрат на автоматизированное тестирование по данной формуле



Используя вышеописанные формулы можно рассчитать величину вложений, нужных для реализации и поддержки автоматизированного и ручного тестирований.

Для автоматизированного тестирования необходимы затраты в 40 158 рублей.

Для ручного тестирования – 70 119 рублей.

Таким образом, автоматизировав всего 2 теста, была получена выгода почти в два раза, при этом, можно ожидать, что последующая автоматизация тестирования приведет к еще большей разнице в эффективности, поскольку этап разработки промежуточного слоя уже пройден (хотя, однозначно по нему потребуются доработки при использовании в реальной компании).

Такая большая разница в эффективности очень показательна, однако нужно понимать, что в зависимости от конкретного проекта и рассматриваемого типа тестирования улучшение эффективности за счет автоматизации может меняться, и в ряде случаев автоматизация может оказаться нецелесообразной.

# Заключение

Сегодня тестирование является неотъемлемой частью процесса производства программных продуктов. Качественное тестирование помогает своевременно выявлять и исправлять ошибки, тем самым уменьшая риски и затраты на разработку программного обеспечения. При автоматизации тестирования скорость и качество тестирования может повыситься, что приведет к еще большему снижению издержек и повышению качества.

В данной работе была проанализирована эффективность внедрения автоматизированного тестирования в компании «ФК Открытие». Для достижения поставленной цели были решены такие теоретические задачи, как описание теоретических основ тестирования, классификация и описание различных видов тестирования, описание методологий, анализ процесса тестирования, выявление и описание критериев корректного построения процесса тестирования. В работе также были определены критерии эффективности процесса тестирования, описаны и проанализированы формулы, позволяющие выразить эффективность данного процесса в денежном эквиваленте.

В практической части работы на примере компании «ФК Открытие» была просчитана эффективность внедрения автоматизированного тестирования для проведения регрессионного тестирования. Для подсчета эффективности были применены формулы, описанные ранее. После полученных результатов, в которых говорилось о целесообразности применения автоматизированного тестирования в теории, были автоматизированы два больших теста с целью подтвердить или опровергнуть полученные ранее результаты.

В результате автоматизации двух тестов были получены результаты, подтверждающие целесообразность внедрения автоматизированного тестирования для проведения регрессионного тестирования.

Данные результаты показывают целесообразность только для конкретной компании и для конкретного вида тестирования. Необходимо понимать, что каждая компания сама должна оценивать целесообразность внедрения автоматизированного тестирования программного обеспечения.

Для правильного построения и автоматизации процесса тестирования необходимо обладать теоретическими знаниями в данной области, различать виды тестирования, понимать, как именно должен быть написан правильный тест-кейс, для повышения качества тестирования программного обеспечения. Данная работа актуальна для компаний, нацеленных на развитие направления тестирования, автоматизацию процесса тестирования и повышение качества своего продукта, поскольку в работе описаны необходимые теоретические основы, более того, проанализированы критерии эффективности процесса тестирования и на конкретном примере рассчитана эффективность внедрения автоматизированного тестирования в компании.

# Список литературы

1. Джек Фолк, Сэм Канер, Енг. Кек Нгуен. Тестирование программного обеспечения. Издательство ДиаСофт, 2001.

2. Савин Роман. Тестирование DOT COM. Издательство Дело, 2007.

3. Виды Тестирования [Электронный ресурс]/ Про Тестинг - Тестирование Программного Обеспечения. URL: http://www.protesting.ru/testing/types/sanity.html

4. Certifying Software Testers Worldwide [Электронный ресурс]/ URL: http://www.istqb.org/

5. Александр Хрущев. Эффективность использования автоматических тестов в ИТ-проектах. Доклад на конференции CEE-SECR 2009, октябрь 2009.

6. Оценка эффективности автоматизации тестирования [Электронный ресурс]/ Технологии качества. URL: http://a1qa.ru/blog/otsenka-effektivnosti-avtomatizatsii-testirovaniya/

7. Гребенюк В. М. Oценка целесообразности внедрения автоматизированного тестирования. Институт Государственного управления, права и инновационных технологий (ИГУПИТ). Интернет-журнал «НАУКОВЕДЕНИЕ» №1 2013

8. Джефф Рэшка, Элфрид Дастин, Джон Пол. Автоматизированное тестирование программного обеспечения. Внедрение, управление, эксплуатация. Издательство Лори, 2012.

9. Ron Patton. Software Testing. 2005.

10. Винниченко И.В. Автоматизация процессов тестирования. Издательство Питер, 2005.

11. Рекс Блек. Ключевые процессы тестирования - М.: Издательство Лори, 2014. – 544 с.

12. Сертификация программного обеспечения (ПО) [Электронный ресурс]/ Национальная сертификационная палата. URL: http://www.nspru.ru/sertsoftware/

13. Автоматизированное тестирование [Электронный ресурс]/ GitHub. URL: https://gist.github.com/codedokode/a455bde7d0748c0a351a

14. Основные положения тестирования [Электронный ресурс]/ Интересные публикации / Хабрахабр. URL: https://habrahabr.ru/post/110307/

15. Что такое Конфигурационное тестирование [Электронный ресурс]/ software-testing. URL: http://software-testing.org/testing/chto-takoe-konfiguracionnoe-testirovanie-configuration-testing.html

16. Автоматизация тестирования [Электронный ресурс]/ Перфоманс Лаб. URL: http://www.performance-lab.ru/avtomatizacija-testirovanija