Министерство науки и высшего образования РФ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рыбинский государственный авиационный технический университет

имени П.А. Соловьева»

Факультет радиоэлектроники и информатики

Кафедра МПО ЭВС

**выпускная квалификационная работа**

Разработка системы автотестирования пользовательских интерфейсов программной линейки продуктов НПО Криста

на соискание степени бакалавр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по направлению 09.03.04 Программная инженерия

профиль бакалавриата: Разработка программно-информационных систем\_\_\_\_\_\_\_\_\_\_

**Пояснительная записка**

Соискатель, студент группы ИПБ-17 Бубенцов С.А.

*(Код) (Подпись, дата ) (Фамилия И.О.)*

Руководитель ст.преп. Смирнов Н.В.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

Консультант

по экономике к.т.н., доцент Клементьева Н.А.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

Нормоконтролер ст.преп. Задорина Н.А.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

К защите допустить

Зав. кафедрой к.ф.-м.н., профессор Паламарь И.Н.

*(Уч. степень, звание) (Подпись, дата) (Фамилия И.О.)*

ВКР передана в ГЭК «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Секретарь ГЭК ст.преп. Задорина Н.А.

*(Подпись) (Фамилия И.О.)*

Рыбинск 2021

Оглавление

[Введение 11](#_Toc74660749)

[1 Описание проблемы 12](#_Toc74660750)

[1.1 Обзор предметной области 12](#_Toc74660751)

[1.2 Формулировка проблемы 16](#_Toc74660752)

[1.3 Постановка задачи 17](#_Toc74660753)

[2 Обзор аналогов 19](#_Toc74660754)

[2.1 Определение критериев анализа 19](#_Toc74660755)

[2.2 Сравнительная характеристика существующих разработок 20](#_Toc74660756)

[2.3 IBM Rational Functional Tester 21](#_Toc74660757)

[2.4 HP QuickTest Professional 21](#_Toc74660758)

[2.5 Сравнение существующих систем 22](#_Toc74660759)

[3 Программная документация 24](#_Toc74660760)

[3.1 Техническое задание на программное обеспечение 24](#_Toc74660761)

[3.1.1 Общие сведения 24](#_Toc74660762)

[3.1.2 Назначение и цели создания системы 25](#_Toc74660763)

[3.1.3 Требования к системе 25](#_Toc74660764)

[3.2 Пояснительная записка к программному обеспечению 28](#_Toc74660765)

[3.2.1 Анализ предметной области 28](#_Toc74660766)

[3.2.2 Формулировка проблемы 28](#_Toc74660767)

[3.2.3 Обзор аналогов 28](#_Toc74660768)

[3.2.4 Назначение и область применения 28](#_Toc74660769)

[3.2.5 Технические характеристики 28](#_Toc74660770)

[3.3 Описание программы 37](#_Toc74660771)

[3.3.1 Общие сведения 37](#_Toc74660772)

[3.3.2 Функциональное назначение 38](#_Toc74660773)

[3.3.3 Описание логической структуры 38](#_Toc74660774)

[3.3.4 Используемые технические средства 53](#_Toc74660775)

[3.4 Программа и методика испытаний 55](#_Toc74660776)

[3.4.1 Объект испытаний 55](#_Toc74660777)

[3.4.2 Цель испытаний 55](#_Toc74660778)

[3.4.3 Требования к программе 55](#_Toc74660779)

[3.4.4 Описание испытаний 55](#_Toc74660780)

[4 Эксплуатационная документация на программный продукт 59](#_Toc74660781)

[4.1 Руководство системного администратора 59](#_Toc74660782)

[4.1.1 Общие сведения о программе 59](#_Toc74660783)

[4.1.2 Системные требования 59](#_Toc74660784)

[4.1.3 Установка программы 60](#_Toc74660785)

[4.1.4 Файл конфигурации. Составление и правка 60](#_Toc74660786)

[5 Акт испытаний программного продукта 61](#_Toc74660787)

[6 Экономическое обоснование 62](#_Toc74660788)

[6.1 Экономическое обоснование разрабатываемого программного обеспечения 62](#_Toc74660789)

[6.2 Сравнение с аналогами 62](#_Toc74660790)

[6.3 Описание функций, назначения и потенциальных пользователей программного обеспечений 63](#_Toc74660791)

[6.3.1 Назначение программного обеспечения 63](#_Toc74660792)

[6.3.2 Функции программного обеспечения 63](#_Toc74660793)

[6.3.3 Потенциальные пользователи программного обеспечения 64](#_Toc74660794)

[6.4 Расчет затрат на разработку программного обеспечения 64](#_Toc74660795)

[6.5 Оценка результата от использования ПО 69](#_Toc74660796)

[Заключение 73](#_Toc74660797)

[Список литературы 74](#_Toc74660798)

# Введение

В современном мире, автоматизированное тестирование является важной частью подхода к разработке программного обеспечения. Оно не позволяет полностью исключить ручное тестирование из процесса разработки, но значительно снижает его долю, помогает убрать рутину из процесса тестирования, позволяет значительно сократить время и ресурсы, уменьшить расходы компаний-разработчиков, затрачиваемые на тестирование, снизить риск выпуска некачественного продукта на рынок. Поэтому технологии для автоматизации тестирования набирают все большую популярность среди компаний, связанных с разработкой программных продуктов. Особенно выделяется автоматизация тестирования программных продуктов через графический пользовательский интерфейс, поскольку только в этом случае приложение будет протестировано тем же самым способом, которым оно будет использоваться конечными пользователями.

Одной из главных проблем автоматизированного тестирования является его трудоемкость: несмотря на то, что оно позволяет ускорить выполнение тестов и устранить часть рутинных операций, большие ресурсы могут тратиться на обновление самих тестов. Например, при изменении интерфейса приложения необходимо заново переписать все тесты, которые связаны с обновленными окнами, что при большом количестве тестов может отнять значительное количество ресурсов.

В зависимости от функциональности разрабатываемого приложения, языков программирования и других средств, задача организации системы автоматизированного тестирования может кардинально изменяется, приобретая свои персональные проблемы.

# Описание проблемы

## Обзор предметной области

Тестирование программного обеспечения — это метод проверки того, соответствует ли фактический программный продукт ожидаемым требованиям, не содержит дефектов. Тестирование включает выполнение программных/системных компонентов с использованием ручных или автоматизированных инструментов для оценки одного или нескольких интересующих свойств.

Независимо от того, проводятся ли тестирование программного обеспечения и контроль качества на стороне или производятся внутри компании, это неотъемлемая часть любого производственного процесса. Методологии тестирования программного обеспечения относятся к использованию различных стратегий функционального и нефункционального тестирования для проверки того, что программа соответствует определенным стандартам и все действия могут быть доступны и выполнены без ошибок.

Вот некоторые примеры этих методологий:

* приемочное тестирование пользователей;
* функциональное тестирование;
* регрессионное тестирование;
* модульное тестирование;
* тестирование производительности;
* тестирование графического интерфейса.

Данная дипломная работа посвящена разработке системы автотестирования пользовательского интерфейса.

Графический интерфейс пользователя (GUI) — это уровень представления программного обеспечения, веб-сайта или мобильного приложения. В отличие от старого текстового интерфейса командной строки, графический интерфейс представляет собой современное графическое представление основных функций, которые упрощают взаимодействие обычных пользователей с программными приложениями.

После замены интерфейса командной строки (CLI) графическим пользовательским интерфейсом, разработчики начали сталкиваться с недостатками. Оказывается, опыт использования графического интерфейса зависит от размера экрана устройства, операционной системы, выбранного браузера и т. д.

Тестирование графического пользовательского интерфейса позволяет командам разработчиков убедиться, что пользователи могут одинаково воспринимать все элементы приложения независимо от их платформы или браузера.

Элементы пользовательского интерфейса — это компоненты, которые позволяют пользователю взаимодействовать с программой, а тестирование графического интерфейса — это проверка этих элементов.

Список возможных компонентов обширен, но самые распространенные из них включают:

* строки меню;
* диалоговые окна;
* текстовые поля;
* таблицы;
* окна списков;
* кнопки.

В большинстве проектов разработки используется подход многоуровневой архитектуры. То, что разработчик видит, как единое целое, представляется для пользователя в виде графического пользовательского интерфейса. Даже, если нижние уровни архитектуры работают должным образом, в случаях, когда графический интерфейс не настроен правильно, пользователь не сможет выполнять какие-либо действия на веб-сайте. Помимо обеспечения правильного соединения внутреннего и внешнего кода, вот наиболее распространенные особенности автоматического тестирования графического пользовательского интерфейса:

* тестирование интерфейса с точки зрения пользователя. Проведя тестирование графического интерфейса, вы сможете опробовать самые популярные сценарии поведения пользователей на сайте. Тестировщик также будет лучше понимать, как пользователи разных платформ воспринимают веб-сайт;
* повышение качества продукта. Благодаря тестированию графического интерфейса вы сможете анализировать каждую функцию веб-сайта в контролируемой среде. В конце концов, окончательная версия интерфейса будет надежной и сократит количество отрицательных отзывов;
* проверка соответствия элементов. Проверочное тестирование графического интерфейса пользователя гарантирует, что интерфейс соответствует общепринятым стандартам проектирования и разработки;
* снижение рисков в процессе разработки. Внедрение автоматизированного тестирования графического пользовательского интерфейса во время разработки снижает риск пропуска ошибок в конце цикла. Тестирование всего проекта при написании внутреннего кода может привести к его переписыванию и замедлению процесса;

Существует два способа выполнить тестирование графического интерфейса пользователя – вручную и автоматически.

Тестирование графического пользовательского интерфейса в основном выполняется вручную. Однако, многие разработчики пишут сценарии для автоматического тестирования. Есть несколько преимуществ от использования инструментов непрерывной автоматизации:

* повышенная скорость и эффективность. Скрипт может работать

круглосуточно, что невозможно представить, когда речь идет о ручном тестировании. Между тем, вы можете привлекать тестировщиков к более творческим задачам, где человеческий интеллект имеет решающее значение – таким образом, можно значительно повысить эффективность команды;

* уменьшает погрешность. Человеческая ошибка – важный фактор,

влияющий на ручное тестирование. Сохранять концентрацию и концентрацию при проверке производительности функций веб-сайта в течение нескольких часов вряд ли возможно – именно тогда тестировщики начинают ошибаться. С другой стороны, автоматизация сценариев тестирования графического интерфейса пользователя снижает погрешность, поскольку, пока сценарий написан правильно, результаты тестирования будут соответствовать ему;

* обеспечивает согласованность. Сценарии автоматизированного

тестирования переносимы и могут использоваться повторно. Таким образом, такой подход обеспечивает единообразие и может использоваться для нескольких продуктов;

* удобен при поиске ошибок регрессии. Автоматическое тестирование

позволяет разработчикам расширить спектр тестируемых событий и выяснить, правильно ли работает функция во всех наборах обстоятельств. Если бы такое глубокое исследование проводилось вручную, потребовалось бы намного больше времени и рабочей силы;

* экономит время, рабочую силу и деньги. Автоматическое тестирование

подходит как для небольших, так и для крупных компаний. Первым не пришлось бы содержать большой отдел тестирования, чтобы вывести продукт на рынок, в то время как вторым было бы легче выходить на новые рынки.

## Формулировка проблемы

НПО «Криста» специализируется на разработке автоматизированных информационных систем и комплексных решений.

Каждый отдел имеет отдельный репозиторий для работы над своим проектом (модулем). Также имеются модули, используемые несколькими проектами. Существует и отдел новой технологической платформы, создающий инструменты, используемые большим числом прикладных решений. Написание дипломной работы велось в отделе «Исполнение бюджета». Разработчики данного проекта работают над модулем rebudget, содержащем прикладные решения по автоматизации процесса исполнения бюджета, а также над модулем retools, предоставляющем инструменты для нескольких прикладных решений. Иерархия некоторых модулей и использующих их проектов показана на рисунке 1.1.

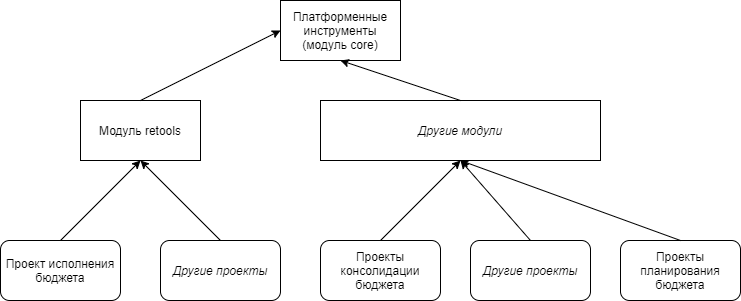


Рисунок .. — Иерархия некоторых модулей и проектов компании «Криста»

Каждый программный продукт компании НПО «Криста» автоматизирует часть бюджетного процесса, который включает в себя этапы составления, утверждения, исполнения, контроля и отчетности. В следствие увеличения количества пользователей программного обеспечения НПО «Криста», а также развития законодательства, увеличивается и количество программных модулей и функциональных возможностей разрабатываемых программных продуктов. При выпуске очередного релиза можно столкнуться с важной проблемой - перестают работать функции приложения, которые всегда были стабильны. К сожалению, часто мы узнаем об этих ошибках после релизов и от клиентов.

На ранних этапах разработки можно обходиться ручным тестированием пользовательского интерфейса по заданному тест-плану. Но с появлением модульной архитектуры, когда несколько команд разработчиков могут вносить свои изменения одновременно, происходит увеличение количества сценариев тестирования. Тестировать их вручную становится все сложнее и сложнее. Автоматизированное тестирование позволило бы решить вышеописанные проблемы.

В данный момент, в прикладных проектах, использующие модуль retools, не проводится автоматизированное тестирование пользовательского интерфейса. В связи с чем появилась необходимость в разработке системы автоматизированного тестирования пользовательского интерфейса этих программных продуктов.

## Постановка задачи

Разрабатываемая система должна быть направлена на проведение автоматизированного тестирования пользовательских интерфейсов программных продуктов, использующих модуль retools, для защиты их от регрессии.

Ключевые особенности разрабатываемой системы автотестирования:

* Предоставление объектов элементов веб-страниц. Для упрощения разработки автотестов, разработчик тестов получает доступ к элементам интерфейса приложения, он может получать информацию об элементах интерфейса и их свойствах, имитировать действия пользователя. В случае изменения структуры любой страницы, не нужно будет заново писать тесты, достаточно изменить код для объектов страниц (только в одном месте).
* Предоставление утилитных классов. Для уменьшения дублирования кода и упрощения чтения разработанных тестов, необходимо разработать вспомогательные классы. Например, они могут использоваться для входа и выхода в систему под конкретным пользователем, поиска окон с ошибками на странице, работы с панелями фильтрации, выполнения скриптов на странице и т.д.
* Предоставление специфических классов исключений. Необходимо разработать модуль, содержащий специфические классы исключений для увеличения понятности для конечного пользователя.
* Формирование графических отчетов по результатам запуска. Система должна предоставлять возможность генерации понятных широкому кругу пользователей графических отчётов по результатам тестирования.

# Обзор аналогов

Есть множество систем для автоматизированного тестирования веб-приложений, однако их основным минусом является стоимость. Даже при скромном функционале и низкой скорости работы они могут иметь высокую стоимостью.

Для сравнения аналогов разрабатываемой системы, необходимо определить критерии сравнения.

## Определение критериев анализа

В качестве критериев для анализа уже существующих систем для автоматизации тестирования будут использоваться:

* распространенность инструмента – подразумевается наличие базы

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

* открытость исходного кода – открытый исходный код, либо

проприетарный формат программного средства. Данный критерий позволяет оценить возможность доработки инструмента для собственных нужд;

* поддержка браузеров – так как рассматривается фреймворк именно для

тестирования веб-приложений, важную роль играет поддержка новых версий популярных браузеров;

* поддержка операционных систем. Поддержка как можно большего

количества операционных систем влияет на выбор того или иного инструмента, в том случае, если тестируемое программное обеспечение (ПО) предполагает мультиплатформенность;

* язык программирования – язык программирования, на котором написана система влияет на простоту/сложность поиска специалистов для разработки и поддержки тестов и самого фреймворка;
* возможности взаимодействия со сторонними библиотеками в рамках Continious Integration. От данного критерия зависит, возможно ли встроить тесты в процесс непрерывной разработки и поставки ПО, что является важным критерием для оценки применимости инструмента.

## Сравнительная характеристика существующих разработок

В данном разделе представлено сравнение разрабатываемой системы

с существующими решениями в данной области по критериям, описанным в предыдущем пункте. Сравнение с аналогами описано в таблице 2.1.

Таблица 2.1 – Сравнение с аналогами

|  |  |  |  |
| --- | --- | --- | --- |
| Категория/Продукт | Разрабатываемая система | IBM RFT | HP QTP |
| Распространенность | Продукт для внутреннего использования | Достаточно распространён | Достаточно распространён |
| Цена | Бесплатный/Внутренние затраты | От 3400$ в месяц | от 600$ в месяц |
| Открытый исходный код | Да | Нет | Нет |
| Поддержка браузеров | Firefox, Edge, IE, Chrome, Opera,  мобильные  браузеры | Firefox, Edge, IE, Chrome, Opera,  мобильные  браузеры | Edge, IE, Chrome, Firefox |

Продолжение таблицы 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Поддержка ОС | Windows, Linux | Windows | Windows |
| Язык программирования | Java | С#, VBScript, Java | С#, VBScript, Java |
| Интеграция с CI | Jenkins | Jenkins, TeamCity | Jenkins, HP Quality Center |

## IBM Rational Functional Tester

IBM RFT — платный инструмент автоматического, функционального и регрессионного тестирования. Это программное обеспечение предоставляет возможности автоматического тестирования для регрессионного, функционального, графического интерфейса и тестирования на основе данных. Особенности инструмента:

* тестирование приложений на Java (J2EE, J2SE, SWT, средства управления AWT/JFC);
* тестирование Web-приложений (HTML, DHTML, XML, JavaScript);
* написание тестовых сценариев на Java;
* упрощение визуализации и редактирования тестов с использованием естественного языка и визуализированных снимков экрана;
* предоставление возможности выполнения одной и той же серии тестовых действий с различным набором тестовых данных;
* поддержка совместного использования функциональных тестов между членами команды и запуска в гибридных средах.

## HP QuickTest Professional

HP QTP — это автоматизированный инструмент функционального тестирования, который помогает тестировщикам выполнять автоматизированные тесты для выявления в тестируемом приложении любых ошибок или дефектов, противоречащих ожидаемым результатам. Он был разработан Mercury Interactive, а затем приобретен HP.

Это инструмент автоматизирует регрессионное и функциональное тестирование приложения.

Особенности инструмента:

* QTP может использовать как технический, так и нетехнический тестировщик. Могут тестироваться как настольные, так и веб-приложения;
* инструмент позволяет тестировать бизнес-процессы;
* разработка автоматизированных тестов с использованием VBSсriрt не требует высококвалифицированного разработчика и относительно проста по сравнению с другими объектно-ориентированными языками программирования;
* легко интегрируется с инструментом управления тестированием (HР-Quality Center), который позволяет легко планировать и контролировать выполнение тестов;
* поддерживает самое большое количество сред разработки программного обеспечения, таких как SAP, Oracle и т. д;
* HP предоставляют полную поддержку для решения технических проблем.

## Сравнение существующих систем

Сравнивая существующие системы, было выявлено, что среди них нет системы, которая бы полностью удовлетворила все потребности в организации процесса автоматизированного тестирования, при этом была бы бесплатной и удобно расширяемой.

Система QuickTest Professional от компании HP является одним из лидеров рынка для создания автотестов, но, к сожалению, имеет очень высокую цену за лицензию в условиях российского рынка IT компаний.

Система Rational Functional Tester обладает схожим функционалом с QuickTest Professional. Так же является платной, стоимость ее лицензии достигает 600 долларов в месяц.

Также были выделены преимущества разрабатываемой системы по сравнению с аналогами:

* условная бесплатность. Плюсом данного подхода, является то, что на разработку будут использованы внутренние ресурсы. Затратами будут только затраты на поддержку и разработку нового функционала, а также начальные затраты на разработку;
* открытый исходный код. Это позволит дорабатывать инструмент каждому отделу для собственных нужд;
* поддержка браузеров. Система будет разработана на основе Selenium Webdriver, который поддерживает как драйверы собственной разработки, так и разработанные любыми сторонними разработчиками. Поэтому система будет поддерживать практически все имеющиеся на рынке браузеры;
* CI/CD. Система разрабатывается в виде отдельного модуля. Это упрощает интеграцию в CI/CD среды, такие как Hudson, Jenkins, и др.

# Программная документация

## Техническое задание на программное обеспечение

### Общие сведения

#### Полное название системы и её условные обозначения

Система автотестирования пользовательских интерфейсов программной линейки продуктов НПО «Криста».

#### Основание для разработки

Основанием для разработки подсистемы оповещений являются задание от подразделения разработки исполнения бюджета компании НПО «Криста», а также приказ № 550-04 от 30.11.2020 на выполнение выпускной квалификационной работы.

#### Порядок оформления и предъявления заказчику результатов работы по созданию системы

Система автотестирования предъявляется в виде программного продукта независимого от существующих модулей компании НПО «Криста» и должна быть, в случае необходимости, использована любым проектом, использующим модуль retools. Заказчик получает модуль с системой и полную документацию к нему в установленные сроки.

Разработанная система должна быть внедрена в проект retools.

#### Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

Разработка технического задания велась согласно документу

ГОСТ 34.602-89.

#### Плановые сроки начала и окончания работы по созданию системы

Плановый срок начала работ по созданию системы автотестирования пользовательских интерфейсов программной линейки продуктов НПО «Криста» – 26 апреля 2021 года.

Плановый срок окончания работ по созданию системы автотестирования пользовательских интерфейсов программной линейки продуктов НПО «Криста» – 10 июня 2021 года.

### Назначение и цели создания системы

#### Назначение системы

Система автотестирования предназначена для автоматизации процесса тестирования пользовательских интерфейсов программных продуктов НПО «Криста».

#### Цели создания системы

Основная цель создания системы автотестирования – защита прикладных проектов от регрессии путём написания автотестов пользовательских интерфейсов.

### Требования к системе

#### Требования к структуре и функционированию системы

В состав системы автотестирования должны входить следующие модули:

* модуль, содержащий классы для работы с элементами веб-страниц приложений;
* модуль, содержащий вспомогательные классы для взаимодействия с элементами веб-страниц приложений;
* модуль, содержащий специфические классы исключений;
* модуль, содержащий классы для работы с инструментами для автоматизации действий веб-браузера Selenium WebDriver и контейнеризации приложений.

Архитектура системы представлена на рисунке 3.1.

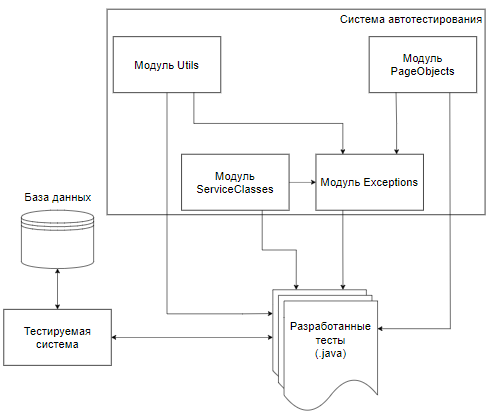


Рисунок . – Архитектура системы

#### Перспективы развития, модернизации системы

Разрабатываемая система автотестирования должна быть готова к добавлению новых объектов веб-страниц, расширению набора методов для работы с элементами страниц, добавлению новых тестовых сценариев.

#### Требования к программному обеспечению системы

При проектировании и разработке системы необходимо максимально эффективным образом использовать ранее закупленное программное обеспечение, как серверное, так и для рабочих станций.

Используемое при разработке программное обеспечение и библиотеки программных кодов должны иметь широкое распространение, быть общедоступными и использоваться в промышленных масштабах.

#### Требования к техническому обеспечению

Добавление системы автотестирования не требует увеличения технических мощностей.

#### Требования к лингвистическому обеспечению

Кодирование системы автотестирования должно происходить на языке java согласно спецификации Java EE. Должна использоваться jdk версии 8. Все комментарии должны вестись на русском языке.

#### Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Использование системы автотестирования не накладывает дополнительных требований на эксплуатацию, техническое обслуживание, ремонт и хранение компонентов системы.

#### Требования к надежности

##### Требования к обеспечению надёжного (устойчивого) функционирования системы

Устойчивое функционирование системы автотестирования должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведён ниже:

* Организация бесперебойного питания сервера, на котором развёрнут тестируемая система.
* Организация технического обслуживания сервера, обеспечивающего бесперебойную работу всех его составляющих (как программных, так и аппаратных).

## Пояснительная записка к программному обеспечению

### Анализ предметной области

Анализ предметной области приведен в разделе 1.1.

### Формулировка проблемы

Формулировка проблемы приведена в разделе 1.2.

### Обзор аналогов

Обзор аналогов приведён в разделе 2.

### Назначение и область применения

Цели, назначение и области использования системы приведены в подразделах 3.1.2.2 и 3.1.2.1 технического задания соответственно.

### Технические характеристики

#### Постановка задачи на разработку программы

Постановка задачи представлена в разделе 1.3.

#### Описание алгоритма и функционирования программы

Система позволяет тестировать, как уже развёрнутый проект, так и с нуля разворачивать самую последнюю сборку нужного проекта в контейнерах.

Общий алгоритм функционирования системы автотестирования представлен на рисунке 3.2.

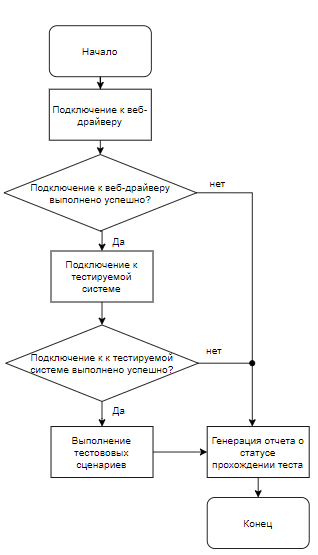


Рисунок .. – Общий алгоритм работы системы автотестирования

Система выполняет подключение к нужному драйверу браузера, программной библиотеке, которая позволяет различным другим программам взаимодействовать с браузером, управлять его поведением. На текущий момент применяется 3 схемы работы с браузерами:

* системный браузер (локальный);
* кластер браузеров;
* тест управляет контейнером с браузером.

Системный браузер – применяется для быстрого запуска тестов на уже развернутом стенде на машине разработчика. При старте, система автотестирования может сама загрузить необходимый драйвер браузера.

Кластер браузеров – промышленное масштабируемое решение для запуска большого количества тестов в большом количестве браузеров. Схема функционирования представлена на рисунке 3.3.

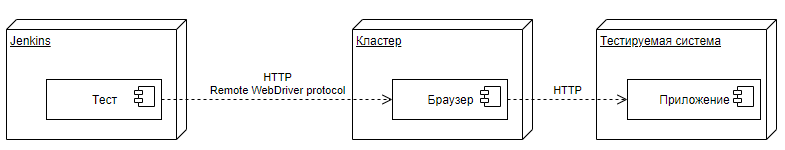


Рисунок .. – Схема функционирования кластера браузеров

Тест управляет контейнером с браузером – до запуска теста поднимаются контейнеры с браузером и тестируемым приложением. Схема функционирования представлена на рисунке 3.4.

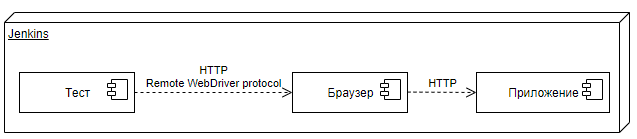


Рисунок .. – Схема функционирования контейнеров с браузером и тестируемым приложением

После успешного подключения к нужному веб-драйверу и к тестируемой системе, запускаются разработанные тесты интерфейса тестируемой системы.

После прохождения всех тестовых сценариев, формируется графический отчёт, содержащий результаты прохождения тестовых сценариев.

При обнаружении ошибок, выполняется рассылка сообщений на электронную почту авторам последних изменений.

Разработанная система предполагает тестирование уже запущенного приложения.

Система поддерживает два режима работы:

* выполнение запуска приложения нужной версии;
* подключение к уже запущенной системе.

При первом режиме работы, перед началом запуска разработанных тестов, необходимо загрузить последний образ тестируемой системы из репозитория. Образ (*Docker-image*) — это файл, включающий зависимости, сведения, конфигурацию для дальнейшего развертывания и инициализации тестируемого приложения. Далее запускается контейнер с этим образом. Контейнер (*Docker-container*) — это исполняемый пакет программного обеспечения, который включает в себя все необходимое для запуска приложения: код, системные инструменты, среду выполнения, системные библиотеки и настройки. Также имеется возможность выполнить подключение развёрнутого приложения к базе данных. Это можно сделать несколькими способами:

* указать путь к нужной копии базы данных;
* указать адрес уже развёрнутой базы данных;
* создать новую пустую базу данных.

После окончания процесса запуска контейнера, приложение будет доступно по URL-адресу, заданному в конфигурационном файле. Алгоритм

функционирования системы при первом режиме работы представлен на рисунке 3.5.

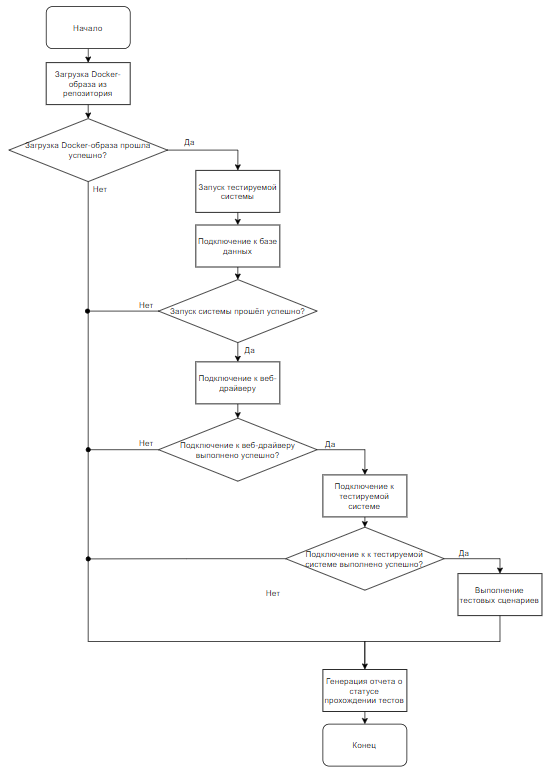


Рисунок .. – Алгоритм работы системы автотестирования (первый режим).

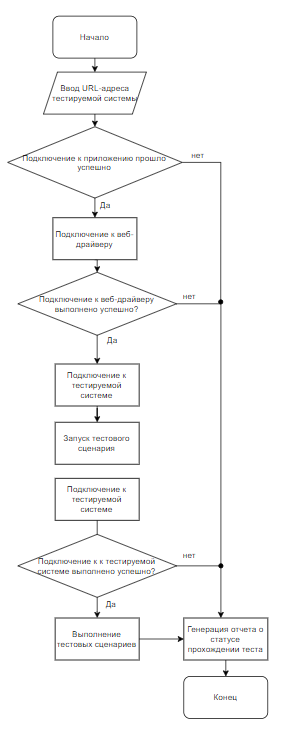
 При втором режиме работы, подключение к уже запущенной системе производится путём задания в конфигурационном файле URL-адреса запущенного приложения. Алгоритм функционирования системы при втором режиме работы представлен на рисунке 3.6.

Рисунок .. – Алгоритм работы системы автотестирования (второй режим)

#### Описание и обоснование выбора состава технических и программных средств

Разработка системы автоматизированного тестирования выполнялась на языке программирования Java.

Java является одним из самых распространенных и популярных языков программирования [7]. Java — это язык программирования высокого уровня, первоначально разработанный Sun Microsystems и выпущенный в 1995 году. Java работает на различных платформах, таких как Windows, Mac OS и различные версии UNIX.

Разработка системы велась для проектов, реализованных на Java, поэтому был выбран именно этот язык программирования.

Для подключения зависимостей и конфигурации системы использовался фреймворк Apache Maven.

Apache Maven — это мощный инструмент управления проектами, основанный на объектной модели проекта. Он используется для сборки проектов, зависимостей и документации. Данный фреймворк используется всеми java проектами компании НПО «Криста».

Для запуска тестов используется фраймворк JUnit. К особенностям данного фраймворка можно отнести:

* открытый исходный код;
* отображение хода выполнения теста;
* предоставление аннотации для определения методов испытаний;
* предоставление методов для тестирования ожидаемых результатов;
* автоматический запуск и предоставление немедленной обратной связи.

Для управления браузером используется инструмент Selenium WebDriver.

Selenium — это библиотека автоматизации браузера. Чаще всего используется для тестирования веб-приложений. Selenium можно использовать для решения любых задач, требующих автоматизации взаимодействия с браузером. Он поддерживает такие браузеры, как Firefox, Chrome, Safari и Internet Explorer. Это позволяет выполнять кросс-браузерное тестирование. Selenium также поддерживает различные языки программирования, такие как C #, Java, Perl, PHP и Ruby, для написания тестовых сценариев. Selenium Webdriver не зависит от платформы, поскольку один и тот же код можно использовать в разных операционных системах, таких как Microsoft Windows, Apple OS и Linux.

Для вывода отчетов в удобной для пользователей форме используется фреймворк Allure.

Allure Framework — это гибкий легкий многоязычный инструмент для создания отчетов о тестировании, который не только показывает краткое представление того, что было протестировано в удобной форме веб-отчета, но и позволяет каждому, кто участвует в процессе разработки, извлекать максимум полезной информации из повседневного выполнения тестов. Данный фреймворк разрабатывался изначально компанией Яндекс, и на текущий момент является свободным ПО с широкими возможностями настройки и привязки практически ко всем доступным инструментам unit-тестов для языка Java: jUnit4, jUnit5, TestNG, Cucumber JVM, Selenide.

Для запуска тестируемой системы используется Docker.

Docker – это открытая платформа для разработки, доставки и запуска приложений. Docker предоставляет возможность упаковывать и запускать приложение в слабо изолированной среде, называемой контейнером. Изоляция и безопасность позволяют запускать множество контейнеров одновременно на одном хосте. Контейнеры легкие и содержат все необходимое для запуска приложения.

Для управления запуском разработанных тестов, используется программная система Jenkins.

Jenkins — это сервер автоматизации с открытым исходным кодом. Jenkins помогает ускорить процесс разработки программного обеспечения за счет его автоматизации. Он управляет и контролирует процессы доставки программного обеспечения на протяжении всего жизненного цикла, включая сборку, документирование, тестирование, развертывание, статический анализ кода и многое другое.

#### Ожидаемые технико-экономические показатели

Внедрение системы автоматизированного тестирования, позволит получить высвобождение ресурсов для отдела тестирования, так как автоматизированные тесты могут исполнятся либо совсем без участия человека, либо с минимальным вмешательством, требуемым для настройки запусков и окружений тестов. Также выигрыш может быть получен за счет увеличения скорости прогона автоматизированных тестов и точности полученных результатов.

## Описание программы

### Общие сведения

#### Обозначение и наименование программы

Система автотестирования пользовательских интерфейсов программной линейки продуктов НПО «Криста» имеет следующие атрибуты:

* наименование файла архива: retools-e2e.jar;
* исходное имя файла: retools-e2e.jar;
* внутреннее имя файла: retools-e2e;
* название продукта: retools-e2e;
* идентификатор группы Maven: ru.krista.retools.retools-e2e;
* идентификатор артефакта Maven: retools-e2e;
* версия 1.00-SNAPSHOT.

#### Программное обеспечение, необходимое для функционирования программы

Системные программные средства, используемые системой автотестирования пользовательских интерфейсов, должны быть, представлены одной из следующих операционных систем: Windows 7 или выше, Ubuntu 16.04 LTS или выше, UbuntuServer 16.04 LTS или выше.

Так же для функционирования программного продукта необходимо следующее предустановленное программное обеспечение стороннего разработчика:

* Oracle JDK версии 8 и выше;
* Docker версии 18 и выше.

#### Языки программирования, на которых написана программа

Исходным языком программирования системы является Java.

### Функциональное назначение

#### Классы решаемых задач

Система автотестирования пользовательских интерфейсов программных продуктов НПО «Криста» - программное решение, обеспечивающее защиту проектов от регрессии, предоставляя работникам предприятия средство для тестирования веб-интерфейсов программных продуктов.

#### Назначение программы

Разрабатываемый программный продукт позволит повысить качество тестирования пользовательских интерфейсов, ускорит процесс тестирования пользовательских интерфейсов, что в свою очередь приведёт к ускорению нахождения и исправления ошибок, связанных с функционированием программного продукта.

### Описание логической структуры

#### Алгоритм программы

Алгоритм работы программы приводится в пояснительной записке раздел «3.2.5.2 Описание алгоритма и функционирования программы».

#### Разработанные модули

##### Модуль «PageObjects»

Для упрощения написания и поддержки тестов веб-интерфейсов программных продуктов НПО Криста, был разработан модуль «PageObjects».

Модуль содержит классы для работы с элементами страниц проектов, использующих модуль retools. Каждый элемент веб-страницы можно описать в виде объекта класса. Взаимодействие пользователя описываются в методах класса, а в тестах остается только бизнес-логика. Данный подход помогает избежать проблем с тестами при изменении верстки веб-приложения. Вам необходимо поправить только класс, описывающий изменённый элемент. Диаграмма классов представлена на рисунке 3.7

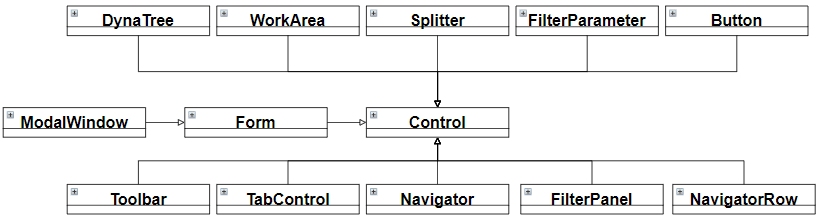


Рисунок .. – Диаграмма классов модуля «PageObjects»

###### Класс «Control»

Базовый класс для работы с элементами страниц (Рисунок 3.8).

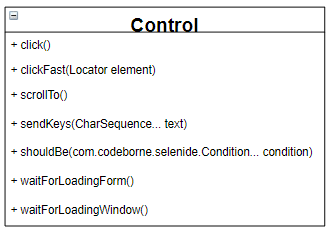


Рисунок .. – Control

Содержит следующие методы:

* click() – производит щелчок мышью по ячейке;
* clickFast(Locator element) – производит быстрый щелчок мыши по заданному элементу;
* pressEnter() – нажимает кнопку «Enter» на текущем элементе;
* scrollTo() – выполняет прокрутку до элемента;
* sendKeys(CharSequence... text) – отправляет последовательность нажатия кнопок на текущий элемент;
* shouldBe(com.codeborne.selenide.Condition... condition) – выполняет проверку условия на элементе;
* waitForLoadingForm() – ожидает, пока не пропадёт окно загрузки формы;
* waitForLoadingWindow() – ожидает, пока не пропадёт окно ожидания операции.

###### Класс «Form»

Базовый класс для реализации форм (Рисунок 3.9).

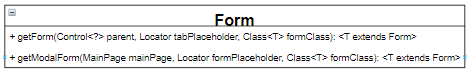


Рисунок .. – Form

Содержит следующие методы:

* getModalForm(MainPage mainPage, Locator formPlaceholder, Class<T> formClass) – получает кастомизированную модальную форму заданного класса;
* getForm(Control<?> parent, Locator tabPlaceholder, Class<T> formClass) – получает кастомизированную немодальную форму заданного класса.

###### Класс «ModalWindow»

Класс для работы с модальным окном (Рисунок 3.10).



Рисунок .. – ModalWindow

Содержит следующие методы:

* close() – отклоняет действие;
* ok() – подтверждает действие.

###### Класс «Splitter»

Класс для работы с компонентом – разделителем (Рисунок 3.11).

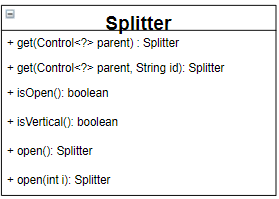


Рисунок .. – Splitter

Содержит следующие методы:

* get(Control<?> parent) – получает ближайший компонент – разделитель;
* get(Control<?> parent, String id) – получает компонент - разделитель по его id;
* isOpen() – проверяет, что первый разделитель находится в раскрытом состоянии;
* isVertical() – флаг показывающий, что данный разделитель – вертикальный;
* open() – открывает первый закрытый разделитель;
* open(int i) – открывает закрытый разделитель с заданным индексом (начиная с 0);

###### Класс «WorkArea»

Класс для работы с рабочей областью (Рисунок 3.12).

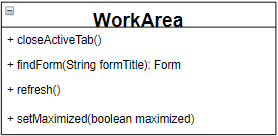


Рисунок .. – WorkArea

Содержит следующие методы:

* refresh() – осуществляет обновление активной формы;
* findForm(String formTitle, Class<? extends T> formClass) – ищет открытую форму не дожидаясь её полной загрузки;
* closeActiveTab() – закрывает активную форму;
* setMaximized(boolean maximized – разворачивает активную форму.

###### Класс «FilterPanel»

Класс для работы с панелью фильтрации (Рисунок 3.13).

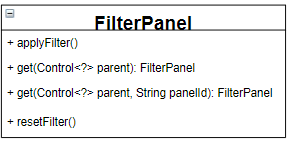


Рисунок .. – FilterPanel

Содержит следующие методы:

* applyFilter() – осуществляет нажатие на кнопку применения фильтра;
* get(Control<?> parent) – получает первую/единственную панель фильтрации;
* get(Control<?> parent, String panelId) – получает панель фильтрации по её идентификатору;
* resetFilter() – осуществляет нажатие на кнопку сброса фильтра.

###### Класс «FilterParameter»

Класс для работы с параметром панели фильтрации (Рисунок 3.14).

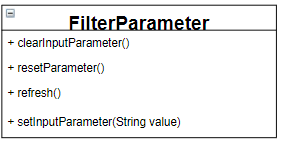


Рисунок .. – FilterParameter

Содержит следующие методы:

* clearInputParameter() – осуществляет очистку значения параметра;
* resetParameter() – осуществляет нажатие на кнопку сброса параметра в фильтре;
* setInputParameter(String value) – устанавливает значение параметра фильтра.

###### Класс «Navigator»

Класс для работы с навигатором рабочих мест (Рисунок 3.15).

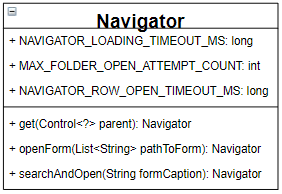


Рисунок .. – Navigator

Содержит следующие константы:

* MAX\_FOLDER\_OPEN\_ATTEMPT\_COUNT – максимальное количество попыток открыть папку навигатора;
* NAVIGATOR\_LOADING\_TIMEOUT\_MS – таймаут ожидания загрузки содержимого навигатора с момента скрытия глобального окна загрузки;
* NAVIGATOR\_ROW\_OPEN\_TIMEOUT\_MS – время ожидания раскрытия папки навигатора.

Содержит следующие методы:

* get(Control<?> parent) – осуществляет получение объекта навигатора по рабочим местам;
* openForm(List<String> pathToForm) – открывает форму из навигатора рабочих мест;
* searchAndOpen(String formCaption) – ищет и открывает форму через строку поиска.

###### Класс «NavigatorRow»

Класс для работы со строкой навигатора рабочих мест (Рисунок 3.16).

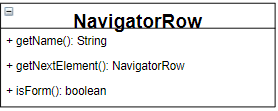


Рисунок .. – NavigatorRow

Содержит следующие методы:

* getName() – получает название строки в навигаторе;
* getNextElement() – получает следующий элемент в навигаторе, после текущего;
* isForm() – признак того что текущая строка в навигаторе ссылка на форму, а не каталог.

###### Класс «DynaTree»

Класс, представляющий собой объект для работы с динамическим деревом (Рисунок 3.17).

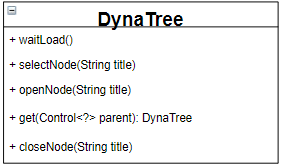


Рисунок .. – DynaTree

Содержит следующие методы:

* waitLoad() – ждет загрузки дерева;
* selectNode(String title) – выбирает узел;
* openNode(String title) – открывает узел;
* get(Control<?> parent) – ищет ближайшее дерево;
* closeNode(String title) – закрывает узел дерева.

###### Класс «TabControl»

Класс для работы с панелью вкладок (Рисунок 3.18).

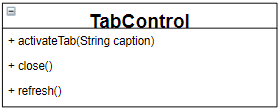


Рисунок .. – TabControl

Содержит следующие методы:

* activateTab(String caption) – активирует вкладку с заданным заголовком;
* close() – закрывает активную вкладку;
* refresh() – производит обновление содержимого активной вкладки.

###### Класс «Toolbar»

Класс для работы с панелью инструментов (Рисунок 3.19).

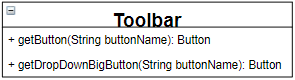


Рисунок .. – Toolbar

Содержит следующие методы:

* getButton(String buttonName) – ищет кнопку;
* getDropDownBigButton(String buttonName) – Ищет кнопку с выпадающим списком.

###### Класс «Button»

Класс для работы с кнопкой на панели инструментов (Рисунок 3.20).



Рисунок .. – Button

Содержит следующие методы:

* getByText(Control<?> parent, String caption) – получает кнопку с панели инструментов по видимому заголовку.

##### Модуль «Utils»

Для уменьшения дублирования кода и упрощения чтения и поддержки тестов, был разработан модуль, содержащий вспомогательные (утилитные) классы. Диаграмма классов представлена на рисунке 3.21.

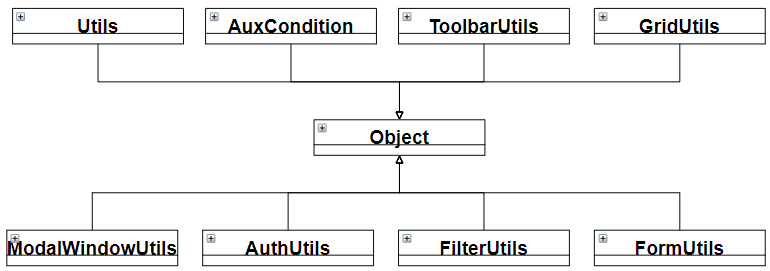


Рисунок . – Диаграмма классов модуля «Utils»

###### Класс «AuthUtils»

Утилитный класс, содержащий методы для входа и выхода в систему под конкретным пользователем (Рисунок 3.22).

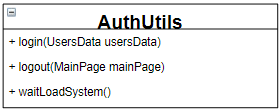


Рисунок .. – AuthUtils

Содержит следующие методы:

* login(UsersData usersData) – осуществляет вход в систему под конкретным пользователем;
* logout(MainPage mainPage) – осуществляет выход из системы;
* waitLoadSystem() – ожидание загрузки системы.

###### Класс «AuxCondition»

Утилитный класс, содержащий вспомогательные условия (Рисунок 3.23).

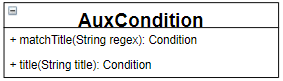


Рисунок .. – AuxCondition

Содержит следующие методы:

* matchTitle(String regex) – условие на соответствие значения атрибута *title* регулярному выражению;
* title(String title) – условие на равенство атрибута *title* ожидаемому значению.

###### Класс «FilterUtils»

Утилитный класс, содержащий различные вспомогательные методы для работы с панелью фильтра (Рисунок 3.24).

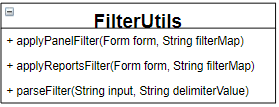


Рисунок .. – FilterUtils

Содержит следующие методы:

* applyPanelFilter(Form form, String filterMap) – применить фильтр на панели фильтра;
* applyReportsFilter(Form form, String filterMap) – применить фильтр по отчету;
* parseFilter(String input, String delimiterValue) – вспомогательный метод для получения фильтра.

###### Класс «FormUtils»

Утилитный класс, содержащий различные вспомогательные методы для работы с формами интерфейсов (Рисунок 3.25).

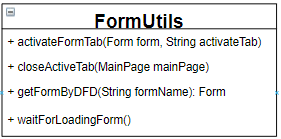


Рисунок .. – FormUtils

Содержит следующие методы:

* activateFormTab(Form form, String activateTab) – переключает на указанную вкладку формы;
* closeActiveTab(MainPage mainPage) – закрывает текущую вкладку;
* getFormByDFD(String formName) – открывает форму с заданным заголовком;
* waitForLoadingForm() – ожидание загрузки формы.

###### Класс «GridUtils»

Утилитный класс, содержащий различные вспомогательные методы для работы с гридами (Рисунок 3.26).

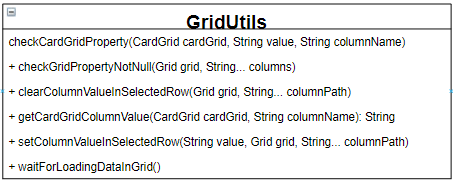


Рисунок .. – GridUtils

Содержит следующие методы:

* checkCardGridProperty(CardGrid cardGrid, String value, String columnName) – проверяет значение ячейки в выбранной колонке;
* checkGridPropertyNotNull(Grid grid, String... columns) – проверяет на нулевое значение ячейки;
* clearColumnValueInSelectedRow(Grid grid, String... columnPath) – очищает ячейку в выбранной строке грида;
* getCardGridColumnValue(CardGrid cardGrid, String columnName) – получает значения ячейки грида;
* setColumnValueInSelectedRow(String value, Grid grid, String... columnPath) – устанавливает указанное значение в текущую ячейку;
* waitForLoadingDataInGrid() – ожидает отображение данных в гриде.

###### Класс «ModalWindowUtils»

Утилитный класс, содержащий различные вспомогательные методы для работы с модальными окнами (Рисунок 3.27).

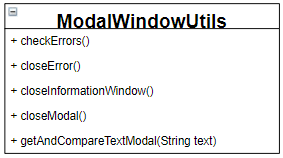


Рисунок .. – ModalWindowUtils

Содержит следующие методы:

* checkErrors() – проверяет наличие модального окна с ошибкой;
* closeError() – закрывает модельное окно с ошибкой;
* closeInformationWindow() – закрывает информационное окно;
* closeModal() – закрывает модальное окно;
* getAndCompareTextModal(String text) – проверяет наличие модального окна с нужным текстом.

###### Класс «ToolbarUtils»

Утилитный класс, содержащий различные вспомогательные методы для работы с панелью инструментов (Рисунок 3.28).



Рисунок .. – ToolbarUtils

Содержит следующие методы:

* findButton(Control<?> control, String title) – выполняет поиск кнопки по заголовку;
* clickButton(Container<?> container, String title) – выполняет нажатие на кнопку.

###### Класс «Utils»

Утилитный класс, содержащий другие вспомогательные методы общего назначения (Рисунок 3.29).

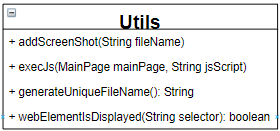


Рисунок .. – Utils

Содержит следующие методы:

* addScreenShot(String fileName) – делает скриншот экрана;
* execJs(MainPage mainPage, String jsScript) – выполняет указанный js скрипт;
* generateUniqueFileName() – генерирует уникальное имя файла по текущим дате и времени;
* webElementIsDisplayed(String selector) – осуществляет проверку видимости элемента.

##### Модуль «Exceptions»

Модуль содержит специфические классы исключений для увеличения понятности для конечного пользователя (Рисунок 3.30. Модуль Exceptions).

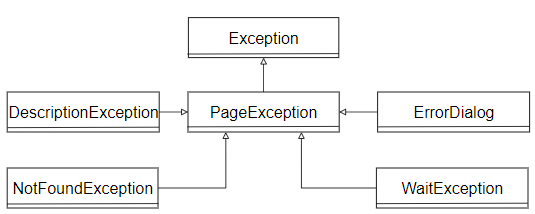


Рисунок .. – Модуль Exceptions

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

* ErrorDialog – исключение, выбрасываемое при непредвиденном появлении окна с ошибкой;
* NotFoundException – исключение, выбрасываемое при не нахождении элемента на веб-странице по определенному локатору;
* DescriptionException – исключение, выбрасываемое при несовпадении данных, ожидаемых от определенного элемента;
* WaitException – исключение, выбрасываемое при истечении времени ожидания;

##### Модуль «ServiceClasses»

Модуль, содержащий классы для работы со следующими инструментами:

* для автоматизации действий веб-браузера Selenium WebDriver;
* для работы с контейнерами при помощи библиотеки TestContainers.

###### Класс «TestContainers»

Данный класс наследуется от класса containers библиотеки для интеграционного тестирования testcontainers-java (Рисунок 3.31).

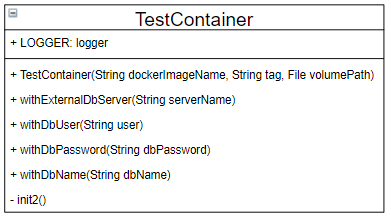


Рисунок .. – TestContainer

Содержит константу – LOGGER. Это переменная класса Logger, относящегося к библиотеке логирования slf4J.

Содержит следующие методы:

* TestContainer(String dockerImageName, String tag, File volumePath) – создаёт контейнер из заданного образа;
* withExternalDbContainer(JavisterPostgreSQLContainer dbContainer) – задаёт использование внешней БД;
* withExternalDbPort – задаёт порт используемой внешней БД;
* withDbUser(String user) – задаёт пользователя БД;
* withDbPassword(String dbPassword) – задаёт пароль БД, к которой должно подключаться приложение;
* withDbName(String dbName) – задаёт имя БД к которой должно подключаться приложение;
* init2() – инициализация контейнера приложения и контейнера БД, если требуется.

###### Класс «TagWebDriver»

Данный класс предназначен для работы с основным инструментарием Selenium WebDriver (Рисунок 3.32).

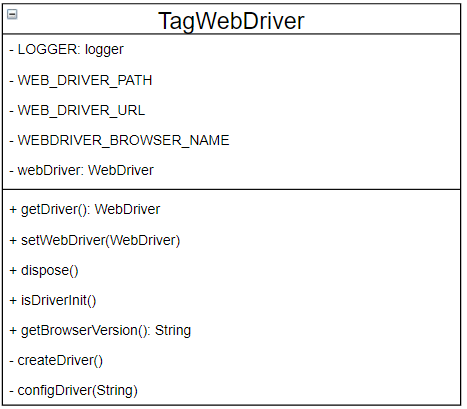


Рисунок .. – TagWebDriver

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

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

### Используемые технические средства

Техническое обеспечение системы должно максимально и наиболее эффективным образом использовать технические средства заказчика.

Минимальные аппаратные средства:

* процессор — двухъядерный с тактовой частотой 2 ГГц или лучше;
* оперативная память — не менее 4 Гб;
* свободное место на жестком диске — не менее 2 Гб;
* дополнительные требования — не менее 2 Гб для файла подкачки.

Стоит отметить, что чем выше производительность рабочей машины, тем выше скорость работы системы в целом.

## Программа и методика испытаний

### Объект испытаний

Объектом испытаний является система автотестирования.

### Цель испытаний

Проверка соответствия разработанной системы техническому заданию, а также выявление сбоев работы системы.

### Требования к программе

Корректное исполнение требований, изложенных в техническом задании.

### Описание испытаний

Проверка корректности работы системы автотестирования выполнялась путём тестирования пользовательского интерфейса системы «Исполнение бюджета». Описание испытаний приведено в таблице 5.1.

Таблица 5.1 – Набор тестов для системы автотестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Проверяемая функциональность | Входные данные | Ожидаемый результат |
| 1 | Подключение к запущенному приложению для тестирования | используемый браузер, URL-адрес развёрнутого приложения | Успешное подключение к приложению, отчёт об успешном прохождении теста |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | Запуск тестируемого приложения в docker-контейнере, подключение к внешней базе данных | Используемый браузер, путь до docker-образа тестируемого приложения, адрес внешней базы данных | Успешный запуск приложения в контейнере, отчёт об успешном прохождении теста |
| 3 | Запуск тестируемого приложения в docker-контейнере с пустой базой данных | Используемый браузер, путь до docker-образа тестируемого приложения | Успешный запуск приложения в контейнере, отчёт об успешном прохождении теста. |
| 4 | Авторизация пользователя в системе | Данные пользователя, используемый браузер, URL-адрес развёрнутой системы | Успешная авторизация в системе, отчёт об успешном прохождении теста |
| 5 | Открытие формы из навигатора | Данные пользователя, используемый браузер, URL -адрес развёрнутой системы, имя формы | Успешное открытие указанной формы, отчёт об успешном прохождении теста |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | Заполнение панели фильтрации в указанной форме | Данные пользователя, используемый браузер, URL -адрес развёрнутой системы, данные для фильтра, название формы, для которой производится фильтрация | Успешная фильтрация данных в указанной форме, отчёт об успешном прохождении теста |
| 7 | Заполнение ячейки таблицы в указанной форме | Данные пользователя, используемый браузер, URL -адрес развёрнутой системы, название формы, данные для занесения в ячейку таблицы, название ячейки таблицы | Успешное занесение данных, в указанную ячейку таблицы, отчёт об успешном прохождении теста |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | Открытие модального окна | Данные пользователя, используемый браузер, URL -адрес развёрнутой системы, название формы, название кнопки, вызывающей открытие модального окна | Успешное открытие модального окна, отчёт об успешном прохождении теста |
| 9 | Переключение между активными информационными формами | Данные пользователя, используемый браузер, URL -адрес развёрнутой системы, названия форм | Переключение между активными информационными формами, отчёт об успешном прохождении теста |
| 10 | Нажатие указанной кнопки на панели инструментов | Данные пользователя, используемый браузер, URL -адрес развёрнутой системы, название формы, наименование кнопки. | Успешное нажатие на указанную кнопку, отчёт об успешном прохождении теста |

# Эксплуатационная документация на программный продукт

## Руководство системного администратора

### Общие сведения о программе

Полное наименование программы: система автотестирования пользовательских интерфейсов программной линейки продуктов НПО Криста.

Краткое название программы: система автотестирования.

Система автотестирования пользовательских интерфейсов программной линейки продуктов НПО «Криста» предназначена для автоматизации процесса тестирования пользовательских интерфейсов программных продуктов НПО «Криста».

Систему автотестирования предполагается использовать для защиты прикладных проектов НПО Криста от регрессии путём написания автотестов пользовательских интерфейсов.

Система автотестирования пользовательских интерфейсов программной линейки продуктов НПО Криста имеет следующие атрибуты:

* наименование файла архива: retools-e2e.jar;
* исходное имя файла: retools-e2e.jar;
* внутреннее имя файла: retools-e2e;
* название продукта: retools-e2e;
* идентификатор группы Maven: ru.krista.retools.retools-e2e;
* идентификатор артефакта Maven: retools-e2e;
* версия 1.00-SNAPSHOT.

### Системные требования

Требования к техническим характеристикам ПК пользователя

* процессор – Intel Pentium 4560 3.5 ГГц;
* объем оперативной памяти – 4 Гб;
* дисковая подсистема – 40 Гб;
* сетевой адаптер – 100 Мбит.

Требования к техническим характеристикам серверов приложений:

* процессор – Intel Xeon 5120 2.2 ГГц;
* Объем оперативной памяти – 32 Гб;
* дисковая подсистема – 1 Тб;
* устройство чтения компакт-дисков (DVD-ROM);
* сетевой адаптер – 100 Мбит.

Требования к техническим характеристикам веб - сервера:

* процессор– AMD Ryzen Threadripper 2950X 3.5 ГГц;
* объем оперативной памяти – 32 Гб;
* дисковая подсистема – 4 Тб;
* сетевой адаптер – 100 Мбит.

### Установка программы

Для того чтобы установить систему, необходимо загрузить и распаковать jar архив.

### Файл конфигурации. Составление и правка

Составление файла конфигурации, а также его правка осуществляется на стороне исполнителя.

# Акт испытаний программного продукта

Объектом испытаний является система автотестирования пользовательских интерфейсов программной линейки продуктов НПО Криста Испытания программного продукта проводили:

* Разработчик Бубенцов С. А.
* Руководитель дипломного проекта Смирнов Н. В.

При этом проверялось соответствие программного продукта техническим требованиям.

Испытания проводились при соблюдении, указанных в техническом задании, требованиях к аппаратуре. Результаты испытаний зафиксированы в протоколах. Испытания проводились по методике испытаний (раздел 3.4).

Проведя тестирование по всем пунктам методики испытаний, можно сказать, что программный продукт работает корректно и удовлетворяет заданным требованиям. Работу можно считать завершенной с положительным результатом.

Студент-дипломник\_\_\_\_\_\_\_\_\_\_Бубенцов С. А.

Руководитель  
дипломного проекта Смирнов Н. В.

# Экономическое обоснование

## Экономическое обоснование разрабатываемого программного обеспечения

С экономической точки зрения внедрение системы автоматизированного тестирования позволит получить высвобождение ресурсов для отдела тестирования, так как автоматические тесты могут прогоняться либо совсем без участия человека, либо с минимальным вмешательством, требуемым для настройки запусков и окружений тестов. Также выигрыш может быть получен за счет увеличения скорости прогона автоматизированных тестов. Плюсом является также высокий уровень повторяемости, то есть обеспечения сравнимых прогонов, исключающих человеческий фактор. Методика расчета экономической эффективности будет включать в себя:

* описание функций, назначения и потенциальных пользователей ПО;
* определение длительности работ и календарного плана. Будет проведена оценка времени, необходимого на проектирование, разработку и внедрение системы;
* сравнение с аналогами;
* расчет расходов на заработную плату работников;
* расчет расходов на сопутствующие материалы;
* расчет расходов на оборудование;
* определение полной себестоимости разработки системы;
* оценка результата от использования ПО.

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

## Сравнение с аналогами

На сегодняшний день, для автоматизированного тестирования, существует множество инструментов, но их основным минусом является стоимость. Для примера будут рассмотрены несколько программных продуктов:

IBM RFT — платный инструмент автоматического, функционального и регрессионного тестирования. Это программное обеспечение предоставляет возможности автоматического тестирования для функционального, регрессионного, графического интерфейса и тестирования на основе данных. Цена составляет от 3400долларов в месяц, что для российского ITрынка является достаточно большой, даже для крупных компаний.

Система HP UFT обладает схожим функционалом с IBM RFT, но также является платной, стоимость составляет от 600 долларов в месяц.

Помимо условной бесплатности, разрабатываемая система обладает такими преимуществами, как наличие открытого исходного кода, и осуществление поддержки внутренним персоналом компании, что увеличит оперативность в решении проблем и в осуществлении необходимых доработок.

## Описание функций, назначения и потенциальных пользователей программного обеспечений

### Назначение программного обеспечения

Система автотестирования предназначена для автоматизации тестирования веб-интерфейса программных продуктов НПО «Криста».

### Функции программного обеспечения

Система обеспечивает возможность выполнения следующих функций:

* подключение к тестируемой системе – необходимо обеспечить

возможность тестирования уже развёрнутого и готового к использованию программного продукта. В данном случае, подключение должно осуществляться, путём ввода URL-адреса системы, которую необходимо протестировать;

* выполнение тестовых сценариев – предоставляется возможность

выполнения разработанных тестов как на локальном браузере, так и на удалённом. Также имеется возможность выполнения разработанных тестовых сценариев по расписанию;

* создание отчётов – в систему интегрирован инструмент построения

графических отчётов автотестов, упрощающий их анализ.

### Потенциальные пользователи программного обеспечения

Пользователями ПО будут являться разработчики, пишущие тестовые сценарии для проверки функциональности тестируемой системы, тестировщики, и менеджеры, анализирующие результаты тестирования.

## Расчет затрат на разработку программного обеспечения

В таблице 6.1 представлены необходимые работы и их длительность.

Таблица 6.1 – Расчет длительности работ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование работ | Длительность работ, дней | | |
| Минимум | Максимум | Ожидаемая |
| 1 | Изучение функциональных требований, присланных заказчиком | 1 | 2 | 2 |
| 2 | Изучение литературы | 2 | 4 | 3 |
| 3 | Разработка алгоритмов | 3 | 10 | 8 | |
| 4 | Разработка ПО | 30 | 50 | 40 | |
| 5 | Отладка ПО | 10 | 20 | 15 | |
| 6 | Тестирование | 5 | 10 | 7 | |

Продолжение таблицы 6.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | Экономическое обоснование | 1 | 2 | 1 |
| 8 | Оформление пояснительной записки | 3 | 9 | 6 |
|  | Итого | 55 | 107 | 82 |

Исходя из данных о работах и их длительности, необходимо рассчитать затраты на разработку ПО. В таблице 2 приводится расчет затрат на основную заработную плату.

Расчет величины основной заработной платы *Зо*, руб. участников команды производится по формуле:

*Зо* = , (6.1)

где n – количество исполнителей, занятых разработкой конкретного ПО;

– часовая заработная плата i-го исполнителя, руб;

– трудоемкость работ, выполняемых i-м исполнителем, ч.

Трудоемкость рассчитывается по формуле:

, (6.2)

где – ожидаемая продолжительность работ;

и – соответственно наименьшая и наибольшая по мнению эксперта длительность работы.

Для разработки любой системы или подсистемы необходимо несколько человек, так как один разработчик со всем справиться не сможет. Поэтому в данном случае в команду входят один программист, один аналитик и один тестировщик.

На примере программиста покажем, как рассчитывается его заработная плата. Трудоемкость работы программиста, равна следующему значению:

= (3 \* 45 + 2 \* 84) / 5 = 60,6 д. = 484,8 ч.

Поскольку месячная заработная плата программиста составляет 80000 рублей, то часовая зарплата составит:

= 80000 / (22⋅8) = 455 руб./ч.

Таблица 6.2. – Расчет затрат на основную заработную плату

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Участник команды | Выполняемые работы | Месячная з/п, р. | Часовая з/п, р. | Трудоем-кость работ, часов | Основная з/п,р. |
| 1 | Программист | 2, 3, 4, 5 | 80000 | 455 | 484,8 | 220 584 |
| 2 | Аналитик | 1, 7, 8 | 55000 | 313 | 65,6 | 20 532 |
| 3 | Тестировщик | 6 | 35000 | 200 | 56 | 11 200 |
| Итого затраты на основную заработную плату разработчиков | | | | | 252 316 | |

Затраты на дополнительную заработную плату разработчика включают выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяются по формуле:

(6.3)

Рассчитаем затраты на дополнительную заработную плату, где Зо – затраты на основную заработную плату с учетом премии; Нд- норматив дополнительной заработной платы, равный 10%.

Социальные отчисления (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле:

, (6.4)

Рассчитаем затраты на социальные отчисления, где – норматив отчислений на социальные нужды (согласно действующему законодательству) равный 30%.

Рассчитаем затраты на эксплуатацию оборудования:

*,*

где – издержки на заработную плату обслуживающего персонала, руб./год;

– годовые издержки на амортизацию, руб./год;

– годовые издержки на электроэнергию, потребляемую ЭВМ, руб./год;

– годовые издержки на вспомогательные материалы, руб./год;

– затраты на текущий ремонт компьютера, руб./год;

– прочие и накладные расходы, руб./год;

в данной формуле принимаем равной нулю.

Затраты на амортизацию определяются по формуле:

*Zам = Cба\* Нам*, (6.5)

где *Cба* – балансовая стоимость компьютера, р.;

*Нам* – норма амортизации, %.

Для вычислительной техники норма амортизации допускается в размере 25 %. *Cба* в данном случае равно 40 000 руб.

Затраты на амортизацию составят:

*Zам* = 40000 \* 0.25 = 10000 рублей / год.

Затраты на электроэнергию, потребляемой за год, определяется по формуле:

*Zэт = Pсп\* Tэф\* Cэт*. \* *A*, (6.6)

где Сэт – стоимость 1 кВт/ч электроэнергии, равная 3,84 руб.;

Тэф – действительный годовой фонд времени работы ЭВМ, час/год;

Pcn – суммарная потребляемая мощность ЭВМ, равная 0,5 кВт;

*A* – коэффициент интенсивного использования мощности машины.

Действительный годовой фонд времени ЭВМ равняется числу рабочих часов для участников команды, за вычетом времени на профилактику и ремонт ЭВМ. Время профилактики: ежемесячная – пять часов, ежегодная – 7 суток. Следовательно, *Тгф* = 260 × 8 – (7 \* 8 + 5 \* 12) = 2 080 – 116 = 1 964 ч. Так как фактическое время работы ЭВМ совпадает с плановым временем работы участников команды, следовательно, коэффициент интенсивного использования мощности машины принимается равным единице.

Таким образом, затраты на электроэнергию за период разработки составят

*Zэт* = 3,84 × 1964 × 0,5 × 1 = 3770,88 руб./год.

Затраты на ремонт ПК, 5% от стоимости ПК

*Zтр* = 40000 × 0.05 = 2000 рублей

Затраты на необходимые материалы, 1% от стоимости ПК

*Zвм* = 40000 × 0.01 = 400 рублей

Прочие затраты на ПК, 5% от стоимости ПК

*Zпр* = 40000 × 0.05 = 2000 рублей

Полные затраты на эксплуатацию составят:

*Zзэ* = 10 000 + 3770,88 + 400 + 2000 + 2000 = 18 170,88 руб./год.

Цену машино-часа можно определить по формуле:

*Смч = Zзэ / Тгф*,

где *Z*зэ – полные затраты на эксплуатацию ЭВМ в течение года;

Тгф – действительный годовой фонд времени работы ЭВМ, час/год.

*Смч* = 18 170,88 / 1 964 ≈ 9,5 руб./ч.

Для расчета фактического времени отладки программы используется формула:

*tфв* = *t*бс + *t*д + *t*о, (6.7)

где tбс – затраты труда на составление программы по готовой блок-схеме, ч;

tд – затраты труда на подготовку документации задачи, ч;

tо – затраты труда на отладку программы на ЭВМ при комплексной отладке задачи, ч.

*tфв=* 484,8 + 65,6 + 56 = 606,4 ч

Тогда затраты на машинное время, составят:

*Zомв* = *Смч* × *t*фв = 9,5 × 606,4 = 5 760,8 руб.

В качестве прочих затрат будем считать плату за отопление, электричество, бумаги и другие внеплановые траты. Размер установим, как 25% от затрат на разработку.

Общие затраты на разработку ПО приведены в таблице 6.3.

Таблица 6.3. – Общие затраты на разработку ПО

|  |  |
| --- | --- |
| Статья затрат | Сумма, рублей |
| Основная заработная плата команды разработчиков(Зо) | 252 316 |
| Дополнительная заработная плата команды разработчиков(Зд) | 25231 |
| Социальные отчисления(Зсоц) | 83 264,34 |
| Машинное время (Zомв) | 5 760,8 |
| Прочие затраты | 91 643 |
| Общая сумма затрат на разработку | 458 215,14 |

Итоговая сумма разработки системы автотестирования составляет 458 215,14 рублей.

## Оценка результата от использования ПО

В результате разработки ПО, заказчик получил продукт, полностью удовлетворяющий его запросам. Использование разработанной системы автотестирования позволяет снизить затраты на поддержку, повысить надежность написанных тестов, а также упросить их написание и чтение.

Благодаря разработанной системе по автоматизации тестирования, время пользовательского интерфейса уменьшилось. Прямым последствием является то, что уменьшилась нагрузка на разработчиков, разрабатывающих автоматизированные тесты. Так же применение разработанной системы для написания автоматизированных тестов позволило упростить анализ отчётов, создаваемых по результатам выполнения тестовых сценариев. Это позволило снизить время реакции на найденные ошибки.

Повышение производительности труда можно рассчитать по формуле:

(6.8)

где Fj – время, которое планировалось пользователем для выполнения работы j-го вида до внедрения программы, ч;

ΔTi - количество часов, которые экономит пользователь.

Если сравнить сколько минут выполняется работа до автоматизации и сколько позже, то можно увидеть сколько экономиться времени и насколько повышается производительность труда.

В таблице 6.4 приведено повышение производительности труда при различных видах работ пользователей.

Таблица 6.4 – Повышение производительности выполнения работ

|  |  |  |  |
| --- | --- | --- | --- |
| Вид работ | До авто­матизации, мин (*Fj)* | Экономия времени, мин. | Повышение про­изводительности труда *Рi*, % |
| Проведение тестирования пользовательского интерфейса | 120 | 100 | 500 |
| Составление отчёта по результатам тестирования | 20 | 15 | 100 |

Из данной таблицы 6.4 видно, что самые трудоемкие виды работ: проведение тестирования пользовательского интерфейса, составление отчёта по результатам тестирования. Автоматизация данных видов работ позволяет сэкономить большое количество времени, и значительно повышает производительность труда.

Среднемесячная зарплата тестировщика компании составляет 35000 рублей, тогда часовая зарплата составит 200 руб./ч. В день он успевает сделать примерно 3 задачи, связанных с проверкой интерфейсных компонентов систем и на каждую задачу у него уходит примерно два часа времени. Следовательно, за выполнение именно этой части работы, его заработок составляет: (200 руб./ч 6 ч) 22 дня = 26 400 руб.

В отделе тестирования системы «Исполнение бюджета» работают 5 тестировщиков. Если предположить, что они будут пользоваться разработанной системой, то экономия, связанная с повышением производительности труда, будет составлять:

руб.

Годовая экономия *Эр* (руб./год) складывается из экономии эксплуатационных расходов и экономии в связи с повышением производительности труда пользователя:

*Эр* = (*Р1* – *Р2*) + Δ*Рп,* (6.9)

где *Р*1 и *Р*2 – соответственно эксплуатационные расходы до и после внедрения разрабатываемой программы;

Δ*Р*п – экономия от повышения производительности труда дополнительных пользователей.

Если эксплуатационные расходы на тестирование пользовательского интерфейса проекта компании составляют примерно 5 000 руб. за один проект в месяц, а таких крупных проектов в месяц 3, то за год ожидается экономия в ((5 000 руб. 3) 12) / 6 = 30000 руб.

Следовательно, годовая экономия *Эр* = 30 000 + 792 000= 822 000 руб./год.

Критерием эффективности создания и внедрения новых средств автоматизации является ожидаемый экономический эффект. Он определяется по формуле:

*Э* = *Эр* – *Ен*×*Кп,,* (6.10)

где Эр – годовая экономия, р.;

Ен – нормативный коэффициент (Eн = 0,15);

Кп – капитальные затраты на проектирование и внедрение, включая первоначальную стоимость программы, р.

Выполним расчет ожидаемого экономического эффекта:

*Э* = 822 000 – 0,15458 215,14 = 753 267,729 руб.

По данному ожидаемому экономическому эффекту видно, что внедрение системы автотестирования благоприятно скажется для предприятия. Сократит количество расходов и при этом цена от внедрения подсистемы окупится в будущем, за счёт экономии средств.

До внедрения данной системы, в ряде проектов, тестирование пользовательского интерфейса, выполнялось вручную. Из-за этого время тестирования значительно увеличивалось, также страдала точность результатов из-за «человеческого фактора».

После автоматизации тестирования пользовательского интерфейса время, требуемое на выполнение работ, значительно сократилось. Прямым последствием является то, что уменьшилась нагрузка на тестировщиков.

Также внедрение разработанной системы, позволило снизить время реакции на найденные ошибки.

# Заключение

В ходе выполнения выпускной квалификационной работы была разработана система автотестирования пользовательских интерфейсов программной линейки продуктов НПО «Криста». Так же была выполнена разработка следующих документов:

* техническое задание (подраздел 3.1);
* пояснительная записка (подраздел 3.2);
* описание программы (подраздел 3.3);
* методика и программа тестирования (подраздел 3.4).

Также, в работе:

* определены критерии для сравнения аналогичных систем автотестирования (подраздел 2.1);
* было произведено сравнение существующих систем (подраздел 2.2).

Система автотестирования предназначается для разработчиков, тестировщиков, менеджеров прикладных проектов НПО «Криста». Она представляет собой инструмент, позволяющий разрабатывать тесты пользовательских интерфейсов.

С экономической точки зрения для обоснования целесообразности разработки и использования системы автотестирования были произведены расчеты эффективности системы, продолжительности и трудоемкости работ при разработке системы, заработной платы разработчика (раздел 6).

Разработанная система успешно прошла ряд испытаний, описанных в методике испытаний, на соответствие требованиям, заявленным в техническом задании. Об этом свидетельствует акт испытаний (раздел 5).

# Список литературы

1. Бек, К. Шаблоны реализации корпоративных приложений / К. Бек. –

М. : Вильямс, 2017. – 176 с.

1. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ. / С. Канер, Д. Фолк, Е. К. Нгуен. – К. : ДиаСофт, 2001. – 544 с.
2. Шаблоны корпоративных приложений / М. Фаулер [и др.]. – М. : Вильямс, 2016. – 544 с.
3. PageObject. Martin Fowler [Электронный ресурс]. – Режим доступа : https://martinfowler.com/bliki/PageObject.html.
4. Apache Cayenne [Электронный ресурс]. – Режим доступа: URL: https://cayenne.apache.org
5. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001 . — 368 с.: ил. (Серия «Библиотека программиста»)
6. Metanit.com [Электронный ресурс]. – Режим доступа: URL: https://metanit.com/java/
7. Allure.qatools.ru [Электронный ресурс]. – Режим доступа: URL: http://allure.qatools.ru
8. Proglib.io [Электронный ресурс]. – Режим доступа: URL: http:// https://proglib.io/p/docker/
9. HP QuickTest Professional [Электронный ресурс] — Режим доступа: URL: https://ru.wikipedia.org/wiki/HP\_QuickTest\_Professional
10. IBM Rational Functional Tester [Электронный ресурс] — Режим доступа: URL: https://en.wikipedia.org/wiki/Rational\_Functional\_Tester
11. jUnit [Электронный ресурс] — Режим доступа: https://ru.wikipedia.org/wiki/JUnit
12. Selenium [Электронный ресурс] — Режим доступа: URL: selenium2.ru
13. Автоматизированное тестирование [Электронный ресурс] — Режим доступа: URL: https://en.wikipedia.org/wiki/Test\_automation
14. Приемочное тестирование или Приемо-сдаточное испытание (Acceptance Testing) [Электронный ресурс] — Режим доступа: URL: http://www.protesting.ru/testing/levels/acceptance.html