Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Школа Информационных технологий и робототехники
Направление подготовки 09.04.02 «Информационные системы и технологии»
Отделение школы (НОЦ) Информационных технологий

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Тема работы
Автоматизация тестирования и мониторинга удалённых серверов с VPN-сервисом

УДК 004.514.054:004.738.5:339

Студент

Группа	ФИО	Подпись	Дата
8ИМ6А	Назимок Павел Павлович		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Шестаков Н.А.	к.т.н.		

КОНСУЛЬТАНТЫ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН	Старикова Е.В.	к.филос.н.		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОКД	Король И.С.	к.х.н.		

ДОПУСТИТЬ К ЗАЩИТЕ:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Профессор ОИТ	Марков Н.Г.	д.т.н., профессор		

Томск — 2018 г.

ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

Код результатов	Результат обучения (выпускник должен быть готов)
Профессиональные и общепрофессиональные компетенции	
Р1	Воспринимать и самостоятельно приобретать, развивать и применять математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте.
Р2	Владеть и применять методы и средства получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе в глобальных компьютерных сетях.
Р3	Демонстрировать культуру мышления, способность выстраивать логику рассуждений и высказываний, основанных на интерпретации данных, интегрированных из разных областей науки и техники, выносить суждения на основании неполных данных, анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями.
Р4	Анализировать и оценивать уровни своих компетенций в сочетании со способностью и готовностью к саморегулированию дальнейшего образования и профессиональной мобильности. Владеть, по крайней мере, одним из иностранных языков на уровне социального и профессионального общения, применять специальную лексику и профессиональную терминологию языка.
Р5	Разрабатывать стратегии и цели проектирования, критерии эффективности и ограничения применимости, новые методы, средства и технологии проектирования геоинформационных систем (ГИС) или промышленного программного обеспечения.
Р6	Планировать и проводить теоретические и экспериментальные исследования в области создания интеллектуальных ГИС и ГИС технологии или промышленного программного обеспечения с использованием методов системной инженерии.
Р7	Осуществлять авторское сопровождение процессов проектирования, внедрения и сопровождения ГИС и ГИС технологий или промышленного программного обеспечения с использованием методов и средств системной инженерии, осуществлять подготовку и обучение персонала.
Р8	Формировать новые конкурентоспособные идеи в области теории и практики ГИС и ГИС технологий или системной инженерии программного обеспечения. Разрабатывать методы решения нестандартных задач и новые методы решения традиционных задач. Организовывать взаимодействие коллективов, принимать управленческие решения, находить компромисс между различными требованиями как при долгосрочном, так и при краткосрочным планировании.
Универсальные (общекультурные) компетенции	
Р9	Использовать на практике умения и навыки в организации исследовательских, проектных работ и профессиональной эксплуатации современного оборудования и приборов, в управлении коллективом.
Р10	Свободно пользоваться русским и иностранным языками как средством делового общения.
Р11	Совершенствовать и развивать свой интеллектуальный и общекультурный уровень. Проявлять инициативу, в том числе в ситуациях риска, брать на себя всю полноту ответственности.
Р12	Демонстрировать способность к самостоятельному обучению новым методам исследования, к изменению научного и научно-производственного профиля своей профессиональной деятельности, способность самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе в новых областях знаний, непосредственно не связанных со сферой деятельности, способность к педагогической деятельности.

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Школа Информационных технологий и робототехники
Направление подготовки 09.04.02 «Информационные системы и технологии»
Отделение школы (НОЦ) Информационных технологий

УТВЕРЖДАЮ:
Руководитель ООП

_____ _____ Марков Н.Г.
(Подпись) (Дата) (ФИО)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

В форме:

магистерской диссертации

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	ФИО
8ИМ6А	Назимку Павлу Павловичу

Тема работы:

Автоматизация тестирования и мониторинга удалённых серверов с VPN-сервисом	
Утверждена приказом директора (дата, номер)	№ 2091/с от 26.03.2018 г.

Срок сдачи студентом выполненной работы:	04.06.18

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе *(наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к особенностям функционирования (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.).*	Объекты исследования: процессы тестирования и разработки удалённых серверов VPN-сервиса и сопутствующего программного обеспечения. Предмет исследования: удалённые сервера VPN-сервиса, SDK-приложение на платформе Android. Документ, описывающий требования: список основных требований от заказчика.
Перечень подлежащих исследованию, проектированию и разработке вопросов *(аналитический обзор по литературным источникам с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе).*	• Анализ подходов и средств реализации автоматизированного тестирования. • Разработка программного решения для автоматизированного тестирования и мониторинга удалённых серверов VPN-сервиса. • Интеграция решения с системами: сбора данных, мониторинга и визуализации данных, непрерывной интеграции. • Анализ и выбор инструмента и паттерна автоматизации тестирования для обеспечения читабельности и поддерживаемости тестов.

	• Реализация автоматизированного тестирования пользовательского интерфейса SDK-приложения. • Внедрение реализованных решений автоматизированного тестирования в процесс разработки VPN-сервиса компанией ООО «Сибэдж».
Перечень графического материала *(с точным указанием обязательных чертежей)*	• Схема работы в сети интернет с использованием VPN-соединения. • Схема тестирования в процессе разработки программного обеспечения в терминах верификации и валидации. • Схема распределения всех составляющих процесса обеспечения качества. • Классическая пирамида автоматизации тестирования. • Схемы ручного и автоматизированного тестирования мобильного приложения. • Схема процесса тестирования удалённых серверов с использованием разработанного фреймворка. • Разработанная статистическая страница в системе визуализации данных. • Список тестов при их запуске в интегрированной среде разработки, в облачной инфраструктуре тестирования приложений.

Консультанты по разделам выпускной квалификационной работы

Раздел	Консультант
Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Старикова Е.В.
Социальная ответственность	Король И.С.

Названия разделов, которые должны быть написаны на русском и иностранном языках:

- Анализ предметной области и постановка задач
- Тестирование программного обеспечения
- Особенности реализации
- Финансовый менеджмент, ресурсоэффективность и ресурсосбережение
- Социальная ответственность

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	29.01.18

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Шестаков Н.А.	к.т.н.		29.01.18

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ИМ6А	Назимок Павел Павлович		29.01.18

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Школа Информационных технологий и робототехники
Направление подготовки 09.04.02 «Информационные системы и технологии»
Уровень образования магистратура
Отделение школы (НОЦ) Информационных технологий
Период выполнения осенний/весенний семестр 2017/2018 учебного года

Форма представления работы:

магистерская диссертация

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	04.06.18

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
04.06.18	Основная часть	75
23.05.18	Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	15
18.05.18	Социальная ответственность	10

Составил преподаватель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Шестаков Н.А.	к.т.н.		29.01.18

СОГЛАСОВАНО:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Профессор ОИТ	Марков Н.Г.	д.т.н., профессор		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	**ФИО**
8ИМ6А	Назимку Павлу Павловичу

Школа	ИШИТР	**Отделение**	ИТ
Уровень образования	Магистратура	**Направление/специальность**	Информационные системы и технологии

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих.	Работа с информацией, представленной в российских и иностранных научных публикациях, аналитических материалах, статистических бюллетенях и изданиях, нормативно-правовых документах.
2. Нормы и нормативы расходования ресурсов.	
3. Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования.	
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Оценка коммерческого потенциала, перспективности и альтернатив проведения НИ с позиции ресурсоэффективности и ресурсосбережения.	Оценка потенциальных потребителей продукта, анализ конкурентных технических решений, QuaD-анализ, диаграмма Исикавы, SWOT-анализ.
2. Планирование и формирование бюджета научных исследований.	Определение целей и результатов проекта, ограничений и допущений. Планирование этапов работ, исполнителей и затрат на проведение исследования.
3. Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования.	Расчет интегральных показателей эффективности исследования, выбор наилучшего исполнения.
Перечень графического материала *(с точным указанием обязательных чертежей)*:	
1. Оценка конкурентоспособности технических решений. 2. Диаграмма Исикавы. 3. Матрица SWOT. 4. Альтернативы проведения НИ. 5. График проведения и бюджет НИ. 6. Оценка ресурсной, финансовой и экономической эффективности НИ.	

Дата выдачи задания для раздела по линейному графику	29.01.18

Задание выдал консультант:

Должность	**ФИО**	**Ученая степень, звание**	**Подпись**	**Дата**
Доцент отд. социально-гуманитарных наук	Старикова Екатерина Васильевна	к.филос.н.		

Задание принял к исполнению студент:

Группа	**ФИО**	**Подпись**	**Дата**
8ИМ6А	Назимок Павел Павлович		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	**ФИО**
8ИМ6А	Назимку Павлу Павловичу

Школа	ИШИТР	**Отделение**	ИТ
Уровень образования	Магистратура	**Направление/специальность**	Информационные системы и технологии

Исходные данные к разделу «Социальная ответственность»:	
1. Характеристика объекта исследования и области его применения	• Объект исследования: удалённые сервера VPN-сервиса. • Описание разработки: реализация автоматизированного тестирования и мониторинга удалённых серверов VPN-сервисов. • Область применения: информационные технологии. • Рабочая зона: компания ООО «Сибэдж», занимающаяся заказной разработкой программного обеспечения.
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Производственная безопасность	1.1. Анализ вредных и опасных факторов, которые может создать объект исследования. 1.2. Анализ вредных и опасных факторов, которые могут возникнуть на производстве при внедрении объекта исследования. 1.3. Обоснование мероприятий по защите персонала предприятия от действия опасных и вредных факторов.
2. Экологическая безопасность	2.1. Анализ влияния объекта исследования на окружающую среду. 2.2. Анализ влияния процесса эксплуатации объекта на окружающую среду. 2.3. Обоснование мероприятий по защите окружающей среды.
3. Безопасность в чрезвычайных ситуациях	3.1. Анализ вероятных ЧС, которые может инициировать объект исследований. 3.2. Анализ вероятных ЧС, которые могут возникнуть на производстве при внедрении объекта исследований. 3.3. Обоснование мероприятий по предотвращению ЧС и разработка порядка действия в случае возникновения ЧС.
4. Правовые и организационные вопросы обеспечения безопасности	4.1. Специальные (характерные для проектируемой рабочей зоны) правовые нормы трудового законодательства. 4.2. Организационные мероприятия при компоновке рабочей зоны.

Дата выдачи задания для раздела по линейному графику	29.01.18

Задание выдал консультант:

Должность	**ФИО**	**Ученая степень, звание**	**Подпись**	**Дата**
Доцент отделения контроля и диагностики	Король Ирина Степановна	к.х.н.		

Задание принял к исполнению студент:

Группа	**ФИО**	**Подпись**	**Дата**
8ИМ6А	Назимок Павел Павлович		

Реферат

Выпускная квалификационная работа 109 с., 20 рис., 22 табл., 40 источников.

Ключевые слова: автоматизация тестирования, функциональное, регрессионное тестирование, VPN-сервис, Android-приложение.

Объектом исследования являются процессы тестирования и разработки удалённых серверов VPN-сервиса и сопутствующего программного обеспечения. Предметами исследования являются удалённые сервера VPN-сервиса и SDK-приложение на платформе Android.

Цель работы: анализ, реализация и внедрение решений для автоматизации тестирования и мониторинга удалённых серверов VPN-сервиса и сопутствующего SDK-приложения, разрабатываемых компанией ООО «Сибэдж».

В процессе исследования был произведён анализ предметной области, приведено обоснование необходимости реализации автоматизированного тестирования, выполнен анализ подходов и средств реализации автоматизированного тестирования.

В результате исследования был выбран подходящий способ реализации решений для автоматизированного тестирования и мониторинга удалённых серверов VPN-сервиса и для автоматизированного тестирования пользовательского интерфейса SDK-приложения.

Результатом выполнения магистерской диссертации является реализация фреймворка для автоматизированного тестирования удалённых серверов VPN-сервиса, разработка статистической страницы с результатами тестирования в системе мониторинга и визуализации данных, реализация автоматизированного тестирования пользовательского интерфейса SDK-приложения на языке программирования Java.

Степень внедрения: реализованные тестовые решения внедрены и используются в процессах разработки и поддержки VPN-сервиса

Определения, обозначения, сокращения

Определения:

фреймворк (тестирование) — набор взаимодействующих компонентов, облегчающих создание и выполнение автоматических тестов и представление их результатов.

тест-кейс — описание совокупности шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

баг — означает ошибку в программе или в системе, из-за которой программа выдает неожиданное поведение и, как следствие, результат.

Selenium — инструмент для автоматизации действий веб-браузера.

DevOps — набор практик, нацеленных на активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их рабочих процессов друг в друга.

Prometheus — система сбора и хранения системных данных.

Grafana — система мониторинга и визуализации данных.

CircleCI — система непрерывной интеграции.

Репозиторий — место, где хранятся и поддерживаются какие-либо данные (программы, объекты, метаданные и т. п.)

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки, основанный на системе контроля версий Git.

Firebase Test Lab — облачная инфраструктура тестирования приложений.

Docker — ПО для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы.

Обозначения и сокращения:

ПО — программное обеспечение;

ООО — общество с ограниченной ответственностью;

ИТ — информационные технологии;

IT — Information Technology, информационные технологии;

VPN — Virtual Private Network, виртуальная частная сеть;

SDK — Software Development Kit, набор средств разработки;

SSH — Secure Shell, сетевой протокол прикладного уровня;

ISTQB — International Software Testing Qualifications Board, Международный квалификационный совет по тестированию программного обеспечения;

ISO — International Organization for Standardization, Международная организация по стандартизации;

QA — Quality Assurance, обеспечение качества;

UI — User Interface, пользовательский интерфейс;

GUI — Graphic User Interface;

HTML — HyperText Markup Language, язык гипертекстовой разметки;

API — Application Programming Interface, программный интерфейс приложения;

IDE — Integrated development environment, Интегрированная среда разработки.

Содержание

Введение

В настоящее время систематизированное тестирование — неотъемлемый процесс в промышленной разработке ПО. Задача обеспечения качества выпускаемой продукции является одной из наиболее важных в индустрии информационных технологий.

В сравнении с традиционными методами ручного тестирования, автоматизированное тестирование позволяет существенно сократить расходы компаний, занимающихся разработкой ПО, путём экономии времени и ресурсов, затрачиваемых на процесс тестирования, а также повысить качество выпускаемого продукта [1]. Поэтому технологии автоматизации тестирования набирают всё большую популярность среди софтверных компаний, что и определяет актуальность темы магистерской диссертации.

VPN — технология создания безопасного сетевого соединения — используется многими крупными компаниями и правительственными органами для обеспечения безопасности внутренней сетевой инфраструктуры. Кроме того, на сегодняшний день технология VPN популярна и среди рядовых пользователей интернета в целях получения онлайн-конфиденциальности [2].

Для обеспечения непрерывной и качественной работы крупного VPN-сервиса необходимо регулярное проведение тестирования как аппаратного обеспечения, представляющего собой множество распределённых серверов, так и программного обеспечения, позволяющего установить соответствующее сетевое соединение. В частности, одной из разновидностей таких приложений являются клиентские мобильные приложения, позволяющие пользователю настроить VPN-соединение на своём смартфоне.

Целью данной работы является автоматизация тестирования и мониторинга VPN-сервиса посредством разработки фреймворка для тестирования и мониторинга удалённых серверов на языке программирования Python, а также реализация тестов на языке программирования Java для

тестирования пользовательского интерфейса мобильного Android-приложения для создания VPN-соединения.

Заказным проектом по разработке и поддержке описываемого VPN-сервиса занимается компания ООО «Сибэдж». Практическая значимость результата работы в рамках магистерской диссертации: разработанные тестовые решения в настоящий момент используются в процессе тестирования серверов VPN-сервиса и Android-приложения для создания VPN-соединения. Их внедрение позволило сократить расходы компании на тестирование, ускорить время разработки и улучшить качество программного продукта.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧ

1.1 Описание предметной области

Магистерская диссертация посвящена реализации автоматизированного тестирования разрабатываемого коммерческого проекта на позиции тестировщика в компании ООО «Сибэдж».

ООО «Сибэдж» («SibEDGE») — компания по разработке ПО. Основана в 2007 году. Имеет головной офис в Томске, а также офис в Москве и представительство в Сан-Франциско (США). Основные направления работы: заказная разработка (корпоративные системы, мобильные приложения, решения на базе SharePoint); технологический аутсорсинг (выделение команды разработки, тестирования и администрирования) и IT-консалтинг. Компания имеет большой опыт в работе с крупными заказчиками из нефтегазовой, финансовой, телекоммуникационной областей как на российском, так и на зарубежном рынке.

В компании выполняется тщательное тестирование разрабатываемых программных решений, за что отвечает Отдел обеспечения качества. Тестирование ПО выполняется как в рамках заказных проектов, так и в качестве отдельной услуги [13].

1.2 Описание разрабатываемого компанией проекта

В компании на протяжении долгого времени разрабатывается и поддерживается проект по реализации системы — VPN-сервиса.

VPN — технология создания безопасного сетевого соединения — используется многими крупными компаниями и правительственными органами для обеспечения безопасности внутренней сетевой инфраструктуры. Кроме того, на сегодняшний день технология VPN популярна и среди рядовых пользователей интернета в целях получения онлайн-конфиденциальности [2].

Конечный пользователь получает доступ к системе через клиентские приложения, которые позволяют создать VPN-соединение поверх основной

сети интернет на персональном компьютере или мобильном устройстве. Вместе с тем, в рамках данного проекта ведётся реализация наборов средств разработки (SDK) для различных платформ, в частности для платформы Android.

На рисунке 1 показана схема работы в сети интернет через защищённое от третьих лиц VPN-соединение.

Рисунок 1. Схема работы через VPN-соединение

Инфраструктура VPN-сервиса включает в себя более 1000 удалённых серверов, через которые осуществляется установка и использование соединения по технологии VPN, а также ряд упомянутых ранее клиентских приложений. Кроме того, существуют мобильные приложения, предоставляющие пользовательский интерфейс к функционалу, предоставляемому SDK.

На рисунке 2 представлено окно приложения с формой авторизации в сервисе. На рисунке 3 представлена окно с формой получения дополнительных авторизационных данных и информацией о VPN-соединении.

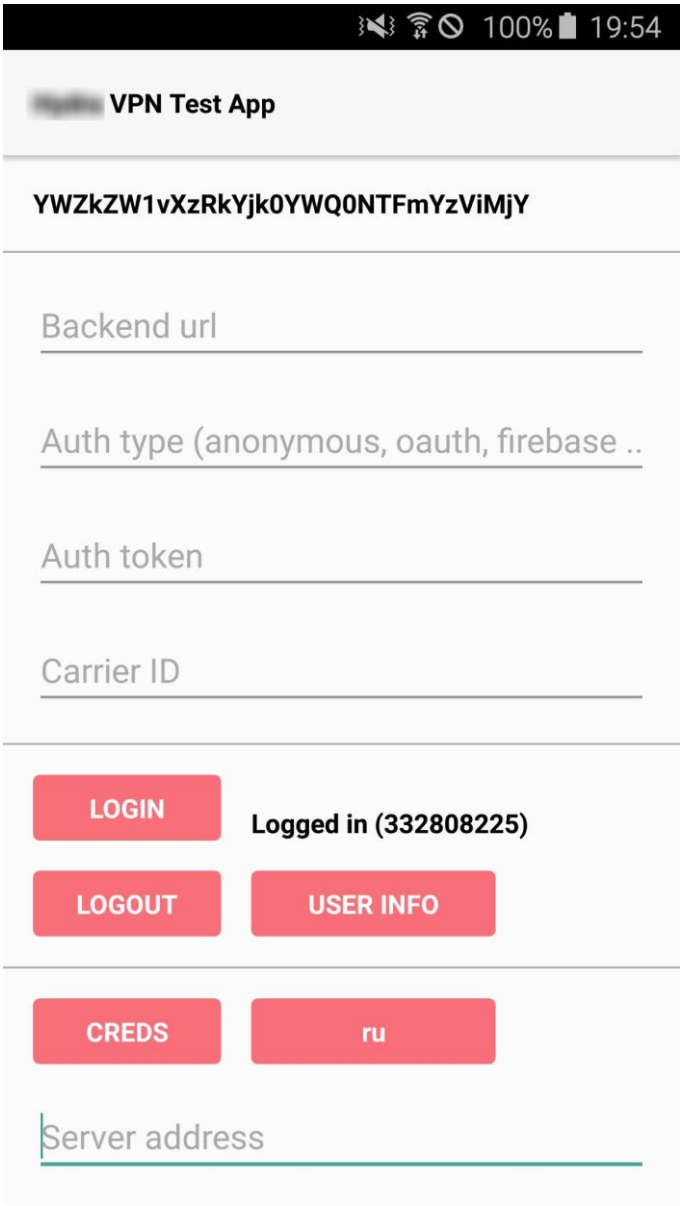

Рисунок 2. Окно приложения. Форма авторизации

Такие SDK-приложения являются простыми с точки зрения пользовательского интерфейса, но имеют гораздо больше функциональных и конфигурационных возможностей по сравнению с обычными пользовательскими клиентскими приложениям. Они используются для демонстрации возможностей пакета SDK потенциальным клиентам-разработчикам ПО, а также для регрессионного тестирования всего VPN-сервиса или отдельных его функций внутри команды разработки системы.

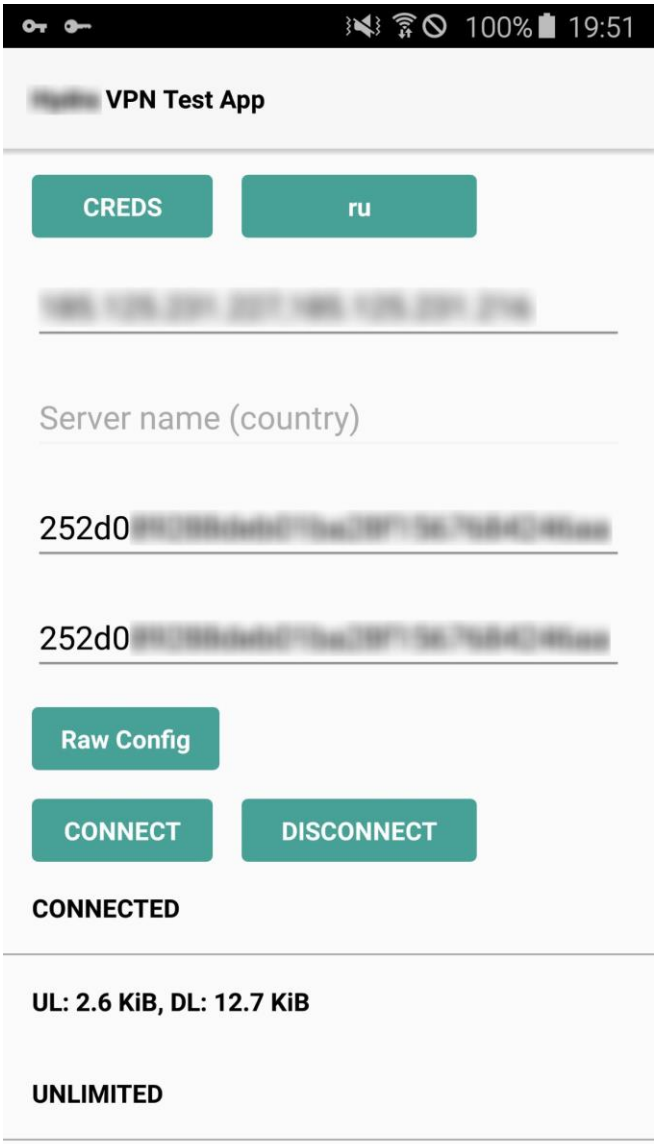

Рисунок 3. Окно приложения. Форма информации о соединении

1.3 Обоснование разработки в рамках магистерской диссертации

Для обеспечения непрерывной и качественной работы описанного VPN-сервиса необходимо регулярное проведение тестирования как аппаратного обеспечения, представляющего собой множество распределённых серверов, так и сопутствующего клиентского ПО.

Магистерская диссертация посвящена решению двух основных задач автоматизации тестирования в рамках проекта разработки VPN-сервиса, которые описаны в разделах 1.3.1 и 1.3.2.

1.3.1 Тестирование и мониторинг удалённых серверов

Существует необходимость разработки программного решения для автоматизации тестирования серверов VPN-сервиса не через клиентские приложения или SDK, а непосредственно с каждого отдельного сервера, минуя пользовательский интерфейс. Такой подход позволяет сымитировать опыт пользователя в сети интернет через VPN-соединение и получить таким образом информацию о нарушениях в работе такого соединения.

В ходе эксплуатации VPN-сервиса специалистами DevOps, занимающимися конфигурацией и обслуживанием серверов, были обозначены наиболее вероятные проблемы при работе через VPN-соединения:

- неправильное определение геолокации со стороны посещаемых электронных ресурсов;
- деградация скорости приёма и передачи данных;
- частичное или полное отсутствие доступа к определённым интернет-ресурсам.

Выявить подобные проблемы на раннем этапе их возникновения достаточно сложно, так как они могут возникнуть на любом из более 1000 серверов системы. Имеют место ситуации, когда о неправильной работе VPN-сервиса сообщают конечные пользователи продукта. Это сказывается на представлении о качестве данного ПО и влияет на его репутацию.

Для того, чтобы предотвращать подобные проблемы на раннем этапе, необходимо автоматизированное решение для регулярной проверки функционирования серверов по обозначенным проблемам, а также организация постоянного мониторинга результатов проведения тестирования. До настоящего момента такая проверка производилась вручную, что при существующем количестве серверов требует больших временных затрат и является практически невыполнимой задачей с точки зрения полного покрытия за один тестовый проход.

1.3.2 Тестирование SDK-приложения

На сегодняшний день набор средств разработки постоянно развивается и совершенствуется. С каждым новым обновлением тестировщику необходимо вручную проверить работоспособность всего приложения, то есть провести регрессионное тестирование. Кроме того, приложение используется и для тестирования определённого функционала всего VPN-сервиса. При этом выполняемые тест-кейсы образуют достаточно рутинный повторяющийся процесс, оставаясь при этом одним из важнейших на каждом текущем этапе разработки.

Чтобы повысить эффективность описанного процесса тестирования, снизить ресурсозатраты и исключить человеческий фактор из рутинного процесса, необходима реализация автоматизированного тестирования пользовательского интерфейса данного SDK-приложения.

1.4 Формирование требований к разработке

Заказчиком разработки, в данном случае компанией ООО «Сибэдж», были поставлены задачи и сформированы требования к программной реализации автоматизированного функционального тестирования выделенных серверов и мониторинга полученных результатов, реализации автоматизированного тестирования пользовательского интерфейса SDK-приложения.

1.4.1 Тестирование и мониторинг удалённых серверов

1. Должно быть разработано программное решение, объединяющее в себе реализацию тестовых сценариев, их запуск на удалённых серверах, сбор и мониторинг полученных результатов тестирования.

2. Должна быть возможность запуска тестов на всех серверах системы без установки дополнительного ПО на каждый сервер (в том числе тестового ПО).

3. Соединение с серверами должно создаваться централизованно из одного места с использованием протокола SSH.

4. Должна быть реализована интеграция с системой сбора данных и оповещения Prometheus.

5. Должно быть реализовано отображение результатов из системы Prometheus в системе мониторинга и визуализации данных Grafana.

6. Запуск тестового приложения должен осуществляться из командной строки с указанием параметров тестирования: список серверов, тестовые сценарии, тип экспорта результатов тестирования.

7. Архитектура разработанного решения должна позволять добавление новых тестовых сценариев, реализовывая при этом только логику самого теста, не затрагивая вспомогательные части кода, отвечающие за: обработку результатов тестирования, интеграцию с системой Prometheus, создание соединения с серверами.

8. Разработанное решение должно быть интегрировано в систему непрерывной интеграции CircleCI с настройкой расписания запуска тестов.

1.4.2 Тестирование SDK-приложения

1. Должно быть реализовано автоматизированное тестирование пользовательского интерфейса приложения путём автоматизации существующих тест-кейсов.

2. Должна быть предусмотрена возможность внедрения тестов в процесс непрерывной интеграции.

3. Должна быть рассмотрена и опробована возможность запуска тестов на облачных устройствах сервиса Firebase Test Lab.

Исходя из предъявленных требований, была произведена постановка цели и задач разработки в рамках магистерской диссертации, которые представлены в разделе 1.5.

1.5 Цель и задачи магистерской диссертации

Анализ, реализация и внедрение решений для автоматизации тестирования и мониторинга удалённых серверов VPN-сервиса и сопутствующего SDK-приложения, разрабатываемых компанией ООО «Сибэдж».

Для достижения данной цели были поставлены следующие задачи:

1. Выявление теоретических основ тестирования, определение места автоматизированного тестирования в общей классификации тестирования.
2. Анализ подходов и средств реализации автоматизированного тестирования.
3. Выбор языков программирования и инструментов для реализации автоматизированного тестирования.
4. Разработка программного решения для автоматизированного тестирования и мониторинга удалённых серверов VPN-сервиса и его интеграция с системами Prometheus и CircleCI.
5. Реализация теста на проверку правильности определения геолокации при использовании поисковой системы Google.
6. Разработка страницы отображения результатов тестирования в системе мониторинга данных Grafana.
7. Реализация автоматизированного тестирования пользовательского интерфейса SDK-приложения для создания VPN-соединения по имеющимся тест-кейсам.
8. Запуск тестов на облачных устройствах сервиса Firebase Test Lab.

2 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 Понятие тестирования

Согласно Международному квалификационному совету по тестированию программного обеспечения (ISTQB) тестирование — это процесс выполнения программы или приложения с целью поиска ошибок ПО [3].

Основными частями процесса тестирования являются верификация и валидация программного продукта. Международная организация по стандартизации (ISO) в стандарте ISO 9000:2015 приводит следующие определения данных терминов:

- Верификация (Verification) — это процесс оценки системы или её компонентов с целью определения того, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. То есть, выполняются ли задачи, цели и сроки по разработке продукта.
- Валидация (Validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе [4].

Другими словами, верификация отвечает на вопрос: «Делаем ли мы продукт *правильно*?», а валидация: «Делаем ли мы *правильный* продукт?»

На рисунке 4 представлена схема тестирования в процессе разработки программного обеспечения в терминах верификации и валидации.

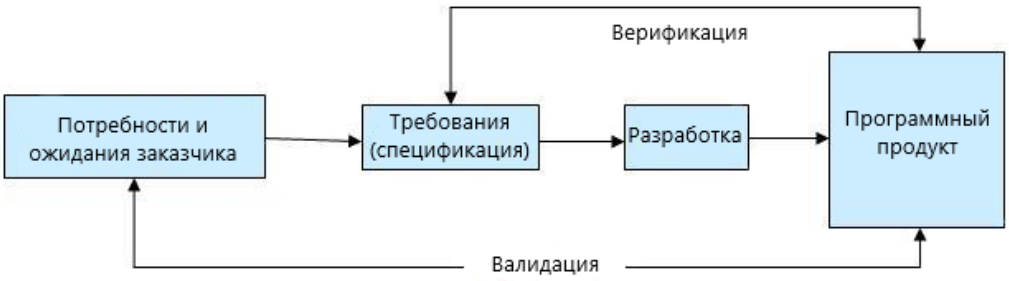

Рисунок 4. Верификация и валидация в процессе разработки ПО

Таким образом, проведение описанных процессов является устоявшимся подходом в ходе реализации ПО. При этом их выполнение может осуществляться разными способами, с применением различных методик и технологических средств. Успешность выполнения зависит от квалификации специалиста и отдела тестирования в целом и позволяет существенно повысить качество итогового программного продукта.

2.2 Обеспечение качества

Согласно стандарту ISO/IEC 25000:2014 качество ПО — это способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям [5].

В настоящее время на рынке ИТ деятельность, связанная с тестированием ПО и соответствующими управленческими процессами, обобщается одним общим понятием — обеспечение качества (Quality Assurance, QA).

Согласно ГОСТ Р ИСО/МЭК 12207-2010 обеспечение качества — это все запланированные и систематические действия, выполняемые в рамках системы качества и продемонстрированные надлежащим образом для обеспечения соответствующей уверенности в том, что объект полностью удовлетворяет требованиям по качеству [6].

В свою очередь, методы обеспечения качества, в частности тестирование, являются частью системы менеджмента качества. Система менеджмента качества включает действия, с помощью которых организация устанавливает свои цели и определяет процессы и ресурсы, требуемые для достижения желаемых результатов [4].

На рисунке 5 представлена схема распределения всех описанных составляющих процесса, направленного на достижение высокого уровня качества разрабатываемого ПО.

Рисунок 5. Менеджмент качества

На сегодняшний день задача обеспечения качества выпускаемой продукции — одна из наиболее важных в индустрии информационных технологий. При этом тестирование является основным способом достижения этой задачи.

Для многих организаций подход к качеству программного обеспечения является одним из способов профилактики: очевидно, гораздо лучше предотвратить возникновение проблем, чем исправить их после. Таким образом, тестирование можно рассматривать как средство предоставления информации о функциональных и качественных атрибутах программного обеспечения, а также как способ выявления сбоев в тех случаях, когда программное предотвращение ошибок не было эффективным [7]. При этом способы осуществления тестирования могут отличаться по уровням, видам, сложности, стоимости, требованиям к квалификации QA-инженера и многим другим характеристикам.

2.3 Классификация тестирования

2.3.1 Уровни тестирования

Тестирование ПО обычно выполняется на разных уровнях в процессе разработки и обслуживания. Уровни определяются в зависимости от объекта тестирования, который может варьироваться: один модуль, группа таких модулей (связанных по назначению, использованию, поведению или структуре) или вся система в целом. Таким образом, три основных этапа тестирования могут быть концептуально выделены в уровни [7], а именно:

1. **Модульный, компонентный (Unit Testing).** На этом уровне выполняется модульное тестирование, которое проверяет функционирование единичных компонентов ПО в изоляции и по отдельности. Реализовывается преимущественно разработчиками, чтобы убедиться, что их код работает правильно и соответствует спецификациям пользователя. Проверке подвергаются отдельные фрагменты кода, образующие единичный компонент системы, такие как классы, функции, интерфейсы и процедуры [3].

2. **Интеграционный (Integration Testing).** На этом уровне выполняется интеграционное тестирование — процесс проверки взаимодействия между программными компонентами. Выполняется, когда два или более модуля интегрированы и образуют рабочий компонент системы, чтобы проверить поведение и функциональность этих модулей после интеграции [7].

3. **Системный (System Testing).** Системное тестирование направлено на проверку поведения целой системы. Большинство функциональных ошибок уже должны быть идентифицированы во время тестирования на модульном и интеграционном уровнях. Системное тестирование подходит для сравнения системы с нефункциональными системными требованиями, такими как безопасность, скорость, точность, надежность и отказоустойчивость. Внешние интерфейсы к другим

приложениям, утилитам, аппаратным устройствам или операционной среде также оцениваются на этом уровне [7].

Ни один из этих этапов не является единственно верным и необходимым для организации процесса тестирования разрабатываемого продукта [7]. Выбор основывается на особенностях конкретной задачи, а также на стандартах качества и процессах его обеспечения внутри компании-разработчика ПО.

Как было отмечено, работа в рамках модульного тестирования выполняется непосредственно разработчиками, в то время как за уровни интеграционного и системного тестирования отвечают QA-специалисты. Поэтому в дальнейшем будут рассматриваться аспекты и особенности выполнения именно второго и третьего уровней.

2.3.2 Виды тестирования

Любое тестирование проводится с учетом конкретной цели, которая заявлена более или менее явно и с различной степенью точности. Указание цели в точном, количественном выражении позволяет установить контроль над процессом тестирования. Тестирование может быть направлено на проверку различных свойств. Тест-кейсы могут быть разработаны для проверки правильности реализации функциональных спецификаций — такое тестирование называется функциональным. Тем не менее, некоторые другие нефункциональные свойства также могут быть протестированы, в том числе производительность, надежность, удобство использования и многие другие. В конечном итоге, объект и вид тестирования зависят от итоговой цели; при этом разные цели решаются на разных уровнях тестирования [7].

Таким образом, можно выделить 2 основных вида тестирования [8].

1. **Нефункциональное тестирование**, куда входит:

- нагрузочное тестирование (Performance and Load Testing);
- стрессовое тестирование (Stress Testing);

- тестирование установки (Installation Testing);
- тестирование удобства пользования (Usability Testing).

Проведение различных видов нефункционального тестирования необходимо далеко не в каждом проекте и зависит о сложности системы, количестве пользователей и методов работы компании-разработчика ПО [8].

2. **Функциональное тестирование (Functional Testing)**. Является основным видом тестирования и реализуется большинством компаний во всех разрабатываемых проектах с целью повышения качества итогового продукта [8].

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом. Оно проверяет реализуемость функциональных требований, то есть способность ПО в определённых условиях решать задачи, нужные пользователям [9].

Данный вид тестирования может выполняться на всех уровнях, описанных ранее. При этом способ выполнения отличается в зависимости от уровня. Так, тестирование в рамках интеграционного уровня может выполняться на специально подготовленных во время процесса разработки тестовых стендах или в тех частях системы, которые отвечают за работу её отдельно взятых компонентов. А тестирование на системном уровне выполняется путём взаимодействия с пользовательским интерфейсом, повторяя действия потенциального пользователя системы.

Успешность и эффективность выполнения любого вида тестирования, в частности функционального, зависит от правильности построения работы всего QA-отдела компании-разработчика, а также от степени автоматизации наиболее важных, типовых или повторяющихся задач тестирования ПО.

2.4 Автоматизация тестирования

В последние годы взгляд на тестирование ПО стал более конструктивным. Тестирование уже не рассматривается как деятельность, которая начинается только после завершения фазы кодирования с ограниченной целью обнаружения багов. Тестирование ПО сегодня — это деятельность, которая должна охватывать весь процесс разработки и поддержки, являясь важной частью фактической конструкции продукта. Действительно, планирование тестирования должно начинаться с ранних этапов жизненного цикла проекта, а планы испытаний и процедуры должны систематически и непрерывно разрабатываться и, возможно, уточняться по мере продолжения разработки [7]. В результате, вероятность выявления потенциальных и фактических недостатков на раннем этапе значительно повышается, что положительно влияет на качество всего продукта, так как ошибка была выявлена разработчиком, а не пользователем в лице заказчика.

Для того, чтобы снизить ресурсозатраты и повысить эффективность процесса тестирования, а следовательно — и самой разработки ПО, применяются различные специализированные методики тестирования, тестировщики работают в тесном контакте с разработчиками, контактируют с заказчиком. Ещё одним способом является замена ручного тестирования на автоматизированное.

2.4.1 Понятие автоматизированного тестирования

На практике тестирование осуществляется путем выполнения определённого набора действий в тестируемом приложении, получении результатов выполнения этих действий и их дальнейшего анализа. Выполняться этот процесс QA-специалистами может как вручную, так и автоматически с использованием различных программных средств. Именно такой процесс верификации ПО, в ходе которого основные функции и шаги теста выполняются автоматически при помощи инструментов для

автоматизированного тестирования, называется автоматизированным тестированием ПО [10].

У автоматизированного тестирования есть как свои преимущества, так и недостатки. К сильным сторонам автоматизированного тестирования относятся:

- быстрая скорость выполнения, намного превосходящая возможности человека;
- отсутствие влияния «человеческого фактора» (невнимательность, усталость);
- возможность многократного и повторного выполнения тестов, что позволяет снизить затраты на процесс тестирования;
- выполнение тест-кейсов высокой сложности, недоступных человеку;
- способность хранить в удобной форме и анализировать большие объёмы тестовых данных.

К недостаткам автоматизированного тестирования относятся:

- необходимость привлечения высококвалифицированного QA-инженера по автоматизации;
- затраты на средства автоматизации, на разработку и сопровождение тестов;
- устаревание тестов в случаях изменений требований, переработки пользовательского интерфейса тестируемых продуктов.

Существуют определённые виды тестирования, которые чаще всего подвергаются автоматизации, делая применение автоматизированного тестирования особенно выгодным.

2.4.2 Применение автоматизированного тестирования

Автоматизация тестирования не позволяет полностью исключить ручное тестирование из процесса разработки, но значительно снижает его долю, уменьшает стоимость и улучшает качество разрабатываемых программных продуктов путём исключения рутинных операций из процесса тестирования.

Можно выделить два наиболее распространённых рутинных процесса тестирования, которые чаще всего автоматизируются QA-специалистами [8].

1. **Дымовое тестирование (Smoke Testing)** — представляет собой тип тестирования ПО, который включает в себя неисчерпывающий набор тестов, направленных на проверку работоспособности наиболее важных функций. Результаты этого тестирования используются, чтобы решить, является ли сборка достаточно стабильной, чтобы продолжить тестирование, или же для проверки правильности функционирования системы [8].

2. **Регрессионное тестирование (Regression Testing)** — собирательное название для всех видов тестирования ПО, направленных на обнаружение ошибок в уже протестированных участках исходного кода. 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 [8].

Сегодня сложно представить себе разработку крупного программного проекта без использования в процессе разработки автоматизированного тестирования. Современный этап развития тестирования характеризуется глубокой интеграцией тестирования с процессом разработки в целом, а также широчайшим использованием автоматизации, колоссальным набором технологий и инструментальных средств для автоматизации [11].

Данная магистерская диссертация посвящена автоматизации дымового и регрессионного функционального тестирования разных частей одной системы на интеграционном и системном уровнях соответственно.

2.4.3 Пирамида автоматизации тестирования

Существует три типа автоматизированных тестов, которые обычно используются в разработке [12].

1. **Модульные тесты (Unit Tests)**. Реализуются разработчиками для проверки отдельных частей программного кода.

2. **Интеграционные, функциональные тесты (Service Tests)**. Тестирование через доступ к функциональному слою, минуя пользовательский интерфейс. Быстрый способ тестирования системных компонентов. Эти тесты могут запускаться без вмешательства пользователя и проверять системные процессы с большой скоростью.

3. **UI-тесты**. Реализуют сквозное (end-to-end) покрытие и обеспечивают стабильную работу системы и удовлетворение критически важных бизнес-требований путём взаимодействия с пользовательским интерфейсом.

Майк Кон организовал эти группы тестов в иерархию, известную как «тестовая пирамида» (рис. 6), которая иллюстрирует соотношение тестов, необходимых для эффективного тестирования системы. На сегодняшний день концепция классической пирамиды является эталонной в сфере автоматизированного тестирования [12].

Рисунок 6. Классическая пирамида автоматизированного тестирования

На вершине пирамиды располагается уровень ручного тестирования, так как оно так или иначе всегда присутствует и выполняется, по крайней мере, для ознакомления с системой и ей обновляемым функционалом.

Стоит отметить схожесть уровневого разделения автоматизированного тестирования с разделением всего тестирования в целом, которое было приведено ранее в разделе 1.3.1. Отличие заключается в том, что при автоматизации системное тестирование выполняется через взаимодействие с пользовательским интерфейсом и сравнение поведения ПО с ожидаемым результатом, а представление уровней в виде пирамиды показывает соотношение количества тестов из каждого уровня соответственно величине блока внутри пирамиды. При этом разделение, предложенное Майком Коном, когда большей частью всех тестов являются unit-тесты, а UI-тесты — меньшей, считается предпочтительным вариантом разработки тестов [12]. Однако это соотношение может меняться в зависимости от конкретной задачи

и особенностей работы команд разработчиков и QA-специалистов отдельно взятой компании-разработчика ПО.

2.5 Инструменты для реализации автоматизированного тестирования

Данная магистерская диссертация посвящена реализации автоматизированных UI-тестов (верхний уровень пирамиды) мобильного приложения на платформе Android, а также решения для выполнения автоматизированных функциональных тестов и сбора их результатов, минуя пользовательский интерфейс (средний уровень пирамиды). Такие виды автоматизации реализуются с использованием специализированных инструментов для автоматизации тестирования, которые предоставляют возможность взаимодействия с интерфейсом тестируемой программы, а также других программных средств, позволяющих использовать низкоуровневые возможности операционных систем или выполнить интеграцию с системами сбора и мониторинга данных.

Существует множество инструментов для реализации автоматизированного тестирования. Каждый из них позволяет решить совершенно разный спектр задач, отличающихся по сложности, объёму, средам исполнения и многим другим характеристикам. Далее приведено описание тех инструментов, которые подходят для решения задач, обозначенных в разделе анализа предметной области.

2.5.1 Тестирование и мониторинг удалённых серверов

Разработка решения для автоматизированного тестирования и мониторинга удалённых серверов включает в себя такие важные детали, как интеграция с системой сбора данных и оповещения Prometheus и создание SSH-соединения с последующим выполнением определённых команд на удалённом сервере. Реализация этих возможностей, с учётом того, что SSH-соединение должно быть установлено с более чем 1000 серверами,

осуществима только на программном уровне. С учётом сопутствующих требований к программному решению, не было найдено готовых инструментов для решения данной задачи. Выходом является разработка собственного решения для тестирования и выполнения всех заявленных требований.

Такие решения обычно называются фреймворками. Существует множество различных определений фреймворка. Наиболее подходящее и широко используемое в сфере автоматизации тестирования: «Фреймворк для автоматизации тестирования представляет собой набор взаимодействующих компонентов, облегчающих создание и выполнение автоматических тестов и представление их результатов» [9]. Другими словами, фреймворк для автоматизации тестирования охватывает всё, что нужно QA-специалисту, чтобы писать, запускать и анализировать автоматизированные тесты, а также может включать в себя реализацию самих тестов.

2.5.2 Тестирование SDK-приложения

В ходе тестирования к приложению обращаются различные инструменты автоматизации. Приложение взаимодействует с пользователем, системой и другими приложениями через такие интерфейсы, как:

- API (Application Programming Interface) — основной интерфейс для взаимодействия с другими программами.
- GUI (Graphic User Interface) — графический интерфейс, используется для взаимодействия с пользователем.
- Net (Networking Interface) — сетевой интерфейс, используется для взаимодействия с интернетом.

Тесты могут использовать все эти интерфейсы для взаимодействия с приложением. При ручном тестировании посредником между тестами и приложением является тестировщик: он преобразует текст тест-кейсов в действия с одним из интерфейсов приложения (рис. 7) [14].

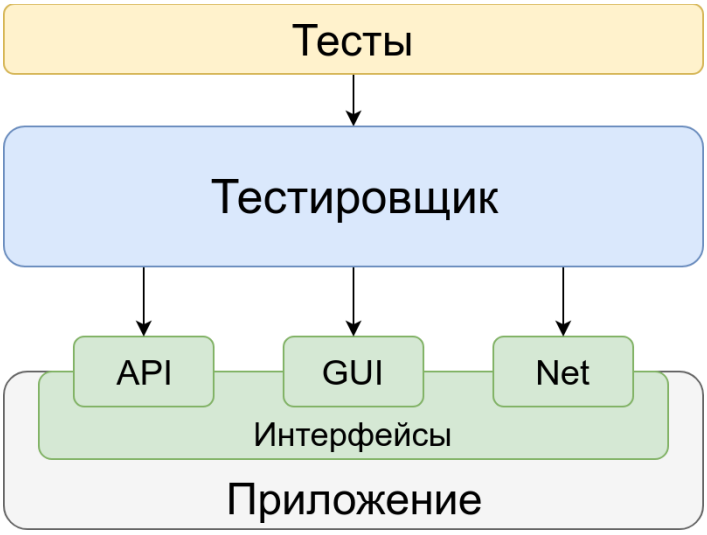

Рисунок 7. Схема ручного тестирования мобильного приложения

Для автоматизации нужно заменить тестировщика на инструменты, которые умеют взаимодействовать с одним или несколькими интерфейсами приложения. Также потребуются утилиты для запуска и формирования набора тестов (рис. 8) [14].

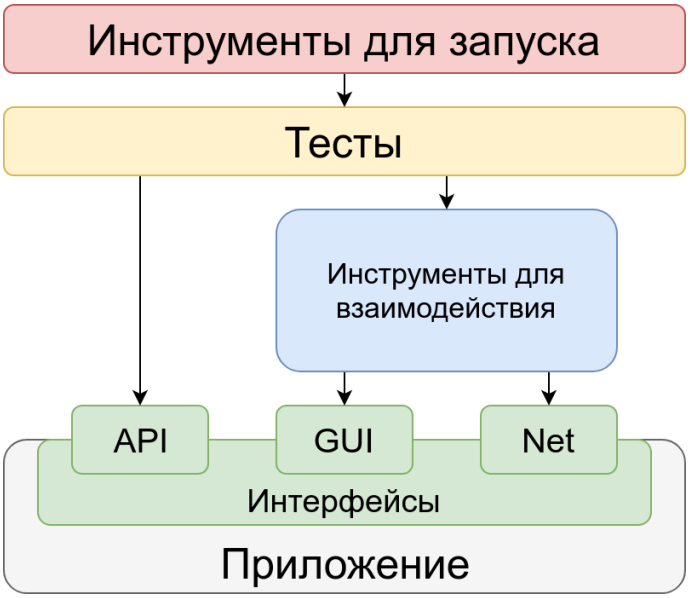

Рисунок 8. Схема автоматизированного тестирования мобильного приложения

Тестируемое SDK-приложение разрабатывается для платформы Android, поэтому далее приведён обзор инструментов, подходящих для

автоматизации тестирования пользовательского интерфейса Android-приложений.

2.5.2.1 Appium

Appium — это бесплатный кроссплатформенный инструмент для мобильной автоматизации с открытым исходным кодом, предназначенный для автоматизации нативных и гибридных мобильных приложений [15]. Преимущества Appium:

- является бесплатным инструментом с открытым исходным кодом;
- поддерживает ряд языков программирования, таких как Java, Python, C#, Ruby, JavaScript, PHP без необходимости внесения изменений в тестируемые приложения;
- работает на различных устройствах и эмуляторах.

Недостатком является то, что Appium не поддерживает обнаружение всплывающих сообщений (Toast). Кроме того, некоторые специалисты по тестированию считают, что полученные тестовые отчеты являются недостаточно информативными [14].

2.5.2.2 Calabash

Calabash — это кроссплатформенный фреймворк с открытым исходным кодом для автоматизации тестирования Android и iOS-приложений. Для каждой ОС есть своя надстройка — Calabash iOS или Calabash Android. Реализация тестирования возможно только на языке Ruby, что является недостатком фреймворка [16].

Инструмент разработан компанией Xamarin, но она прекратила его поддержку в 2017 году, и сейчас он поддерживается только сообществом, вследствие чего теряет популярность на рынке автоматизации тестирования [14].

2.5.2.3 Espresso

Espresso — это бесплатный фреймворк, разработанный компанией Google, с открытым исходным кодом для автоматизации тестирования Android-приложений [17]. Поддерживает языки программирования Java и Kotlin. К преимуществам Espresso относятся:

- наличие рекордера, с помощью которого можно записывать простые сценарии и использовать их на начальном этапе автоматизации;
- реализация полезных функций автоматической синхронизации тестов с пользовательским интерфейсом приложения, что позволяет не писать различные wait-команды (команды ожидания достижения определённого условия);
- простота в настройке и поддержка всех версий Android;
- нет необходимости в отдельном сервере для установки связи между инструментом и тестируемым приложением, как у Appium; запуск производится одновременно с приложением и предоставляет быстрые результаты тестов разработчику [14].

Espresso является частью библиотеки Android Testing Support Library, которая включает в себя инструменты, разработанные компанией Google для тестирования Android приложений. В рамках этой библиотеки реализован класс AndroidJUnitRunner, который позволяет запускать JUnit-тесты на устройствах Android, в том числе с использованием тестовых фреймворков Espresso и UI Automator [14]. JUnit является самым известным, а потому и самым используемым фреймворком для написания и запуска повторяемых тестов на языке Java [18].

Недостатком является то, что Espresso требует доступ к исходному коду для запуска. Соответственно, данный фреймворк не может самостоятельно работать с другими приложениями и системой Android.

Для этих целей Google рекомендует использовать другой их инструмент — UI Automator [17]. Эти два инструмента можно использовать вместе, потому что они — части одной библиотеки. Таким образом, даже в одном тесте можно сочетать команды обоих инструментов.

2.5.2.4 UI Automator

UI Automator — бесплатный фреймворк с открытым исходным кодом, разработанный компанией Google. Поддерживает версии Android начиная с 4.3. Поддерживает языки программирования Java и Kotlin [17]. Преимущества:

- не требует внедрения своего кода в проект, то есть может взаимодействовать с уже скомпилированными приложениями;
- при работе с UI Automator можно использовать возможности системы Android полностью: например, включить геолокацию, вызвать системное приложение, повернуть устройство, нажать на кнопку Home или сделать скриншот.

У UI Automator нет собственного рекордера для тестов, но зато есть утилита UI Automator Viewer, которая позволяет получать данные об элементах запущенного на эмуляторе или реальном устройстве приложения и показывает локаторы этих элементов. Тут же отображается иерархическая структура всех элементов, что очень удобно для их использования в тестах [14].

2.5.2.5 Выводы по обзору инструментов

Наиболее используемым инструментом реализации автоматизированного тестирования как из рассмотренных, так и из всех существующих на данный момент является Appium [14]. Это связано с тем, что он является кроссплатформенным, что означает возможность повторного использования одних и тех же автоматизированных тестов как в приложениях iOS, так и Android. Кроме того, он поддерживает большое количество языков

программирования в сравнении с другими инструментами. Две эти особенности делают его самым популярным средством автоматизации тестирования.

Также, учитывая, что тестируемое SDK-приложение работает на платформе Android, были рассмотрены инструменты, предназначенные исключительно для тестирования приложений, реализованных для операционной системы компании Google. Так фреймворк Espresso, созданный самой компанией Google, учитывает различные особенности Android, что значительно облегчает написание автоматизированных тестов. Однако при использовании его или ему подобных инструментов тестовая база для приложений iOS и Android будет разная, то есть необходимо выполнение двойной работы как в части реализации, так и поддержке автоматизированного тестирования.

Выбор инструмента и обоснование выбора описаны в разделе 3.1.2.

3 ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1 Обоснование выбора инструментов и языков программирования

При выборе инструментов автоматизированного тестирования рекомендуется использовать следующие оценочные критерии [11]:

- поддерживаемые мобильные платформы (распознавание необходимых элементов интерфейса, технологии взаимодействия с ними);
- поддерживаемые типы приложений (веб, гибридные, нативные);
- поддержка запуска тестов на эмуляторах, реальных устройствах;
- IDE (поддерживаемые языки, удобство пользования, инструменты отладки);
- менеджеры тестов (возможности по конфигурированию тестовых наборов, возможность параллельного запуска тестов, настройка параметров запуска);
- создание отчётов.

При выборе инструмента необходимо принимать во внимание специфику приложения, поставленные задачи, квалификацию разработчиков и инженеров по тестированию, технические и финансовые ресурсы, планы на будущее, риски [11].

3.1.1 Тестирование и мониторинг удалённых серверов

Для реализации фреймворка был выбран язык программирования Python. Выбор обусловлен тем, что Python позволяет решать программные задачи любой сложности, обладая при этом простым и удобным синтаксисом. За счёт простоты кода дальнейшее сопровождение программ, написанных на Python, значительно облегчается, что влечёт за собой сокращение расходов и увеличение производительности труда сотрудников с точки зрения бизнеса [19].

Определяющим фактором является то, что для языка Python реализовано множество сторонних модулей, позволяющих решать широкий спектр задач. В частности, модуль для создания неограниченного количества параллельных SSH-соединений.

В качестве IDE была использована PyCharm — интегрированная среда разработки для языка программирования Python, основанная на ПО IntelliJ IDEA от компании JetBrains. Предоставляет средства для анализа кода, графический отладчик, поддержку систем контроля версий.

3.1.2 Тестирование SDK-приложения

Проект VPN-сервиса кроме SDK и клиентских приложений для Android включает в себя также и приложения для iOS. Учитывая этот факт, видится логичным выбор в пользу Appium из-за его кроссплатформенности, что позволило бы иметь и поддерживать только одну тестовую базу вместо двух. Однако над направлениями Android и iOS работают разные команды как разработчиков, так и тестировщиков, поэтому и реализация тестов выполняется каждой командой в рамках своей зоны ответственности.

Выбор был сделан в пользу языка программирования Java и использования комбинации инструментов от компании Google: Espresso, AndroidJUnitRunner и UI Automator. Требование Espresso наличия доступа к исходному коду не стало проблемой, так как реализация тестов выполнялась в основном репозитории приложения, что позволит в будущем внедрить тестирование в процесс непрерывной интеграции без дополнительных усилий. UI Automator делает возможным взаимодействие с такими системными функциями, как, например, настройки интернет и VPN-соединения.

В качестве IDE была использована Android Studio — интегрированная среда разработки для работы с платформой Android, основанная на ПО IntelliJ IDEA от компании JetBrains.

3.2 Реализация фреймворка для тестирования и мониторинга удалённых серверов

Тестирование с помощью реализованного на языке программирования Python фреймворка проходит следующим образом:

1. Тесты запускаются раз в сутки по расписанию, настроенному в системе непрерывной интеграции CircleCI, которая выгружает и строит решение из репозитория в GitHub.
2. На вход поступают параметры тестирования через командную строку: список серверов, набор тестов, способ экспорта данных.
3. Соответственно параметризации запускаются соответствующие тесты параллельно на указанных серверах.
4. Результаты тестирования обрабатываются согласно выбранному способу экспорта данных и отправляются в систему сбора данных Prometheus.
5. Результаты тестирования импортируются из Prometheus в систему мониторинга Grafana, где обрабатываются и отображаются в виде настраиваемых таблиц, графиков и диаграмм.

На рисунке 9 представлена схема процесса тестирования серверов.

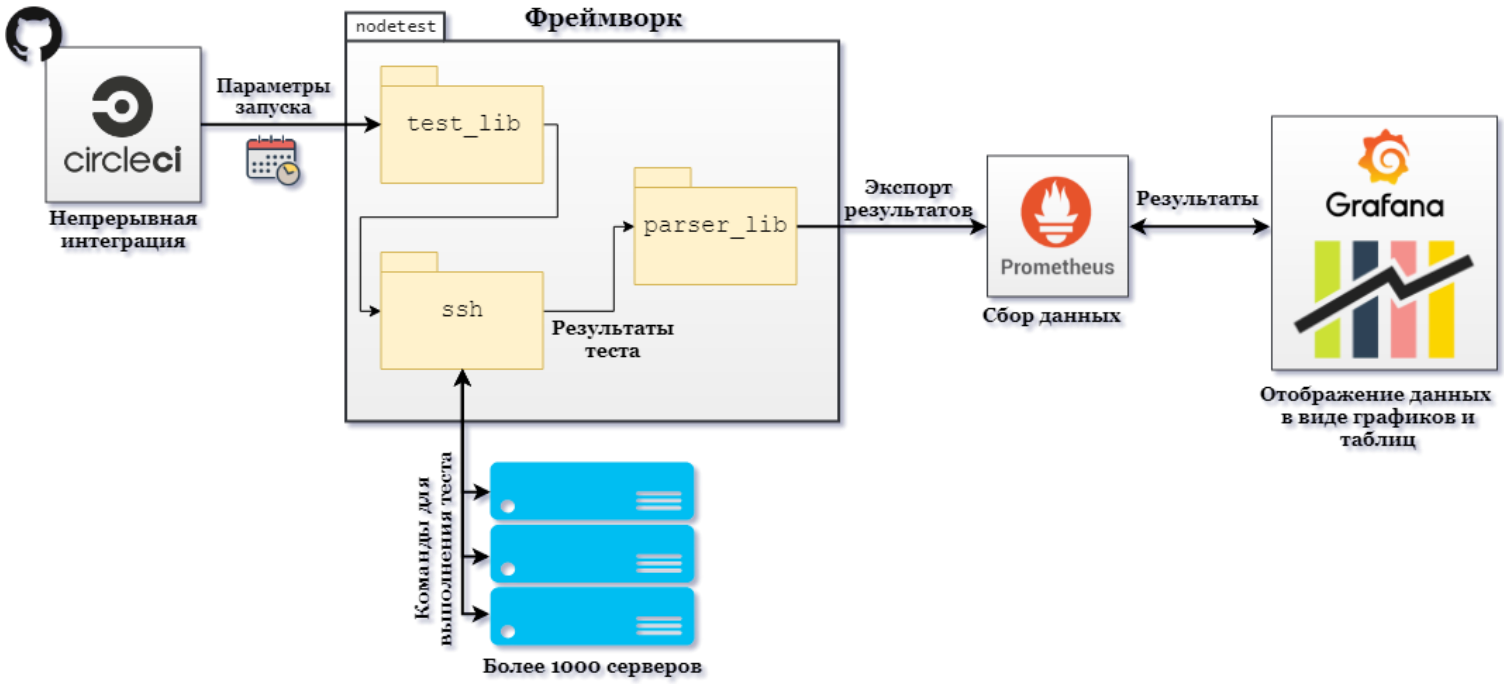

Рисунок 9. Схема процесса тестирования удалённых серверов

Фреймворк разработан таким образом, что в будущем имеется возможность добавлять новые тесты, реализовывая при этом только логику самого теста, не затрагивая вспомогательные части кода, отвечающие за: обработку результатов тестирования, интеграцию с системой Prometheus, создание соединения с серверами, что удовлетворяет одно из требований к разработке, представленных в разделе 1.4.1.

Далее приведены описания основных частей процесса тестирования удалённых серверов.

3.2.1 Параметризация запуска тестов в CircleCI

Репозиторий с исходным кодом фреймворка находится в системе управления версиями GitHub, откуда его забирает система непрерывной интеграции CircleCI, выполняет сборку проекта и запускает тесты.

CircleCI — это современная платформа непрерывной интеграции, выполняющая автоматизированную сборку и выполнение проекта. Одной из самых полезных возможностей, предоставляемых CircleCI, являются готовые образы, запускаемые в среде виртуализации на уровне операционной системы Docker, с различным набором предустановленных операционных систем, средств и языков программирования [20]. Это позволяет не тратить время и ресурсы на построение собственного окружения для запуска собираемого приложения, в данном случае — разработанного фреймворка.

Содержимое конфигурационного файла CircleCI формата YAML, в котором указываются необходимый образ, параметры сборки, команда для запуска тестов и настройка расписания:

```
docker:
    - image: circleci/python:3.6.5-browsers
steps:
    - checkout
    - add_ssh_keys:
        fingerprints:
            - "23:ad:95:44:f3:cb:4a:fe:93:89:79:59:2e:cb:df:57"
    - run:
        name: Install Dependencies
        command: |
            sudo pip install -r requirements.txt
```

```
    - run:
        name: Create the list of servers which will be tested
        command: |
            sudo python nodetest/create_server_list.py --carrier all --
openvpn --ipsec --hydra
    - run:
        name: Run Tests
        command: |
            sudo python nodetest/nodetest.py --server list --test_type
googleredirect --pushgateway
    ...
    - schedule:
        cron: "0 12 * * *"
```

Параметризация запуска тестов выполнена с помощью стороннего Python-модуля argparse. Модуль argparse упрощает создание удобных пользовательских интерфейсов командной строки, позволяя определять наборы входных аргументов, справочные сообщения, а также обрабатывает ошибки неправильного ввода аргументов целевого приложения [21]. Программный код параметризации и запуска тестов, реализованный в модуле nodetest.py:

```
...

if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser()
    arg_parser.add_argument('-s', '--server', nargs='+', default='list',
help='Domains of servers which will be tested')
    arg_parser.add_argument('-t', '--test_type', nargs='+',
choices=['googleredirect', 'googleredirect302', 'alexa'], required=True)
    arg_parser.add_argument('--text', action='store_true', help='Print test
results as text')
    arg_parser.add_argument('--node_exporter', action='store_true',
help='Parse test results for Prometheus node_exporter')
    arg_parser.add_argument('--pushgateway', action='store_true', help='Push
test results to Prometheus using pushgateway')

    args = arg_parser.parse_args()
    ...
    if args.server:
        for test in args.test_type:
            run_test(args.server, test, args.text, args.node_exporter,
args.pushgateway)
    else:
        print('No servers to be tested')
```

3.2.2 Тестовый сценарий

Был создан модуль test_lib.py, который отвечает за реализацию тестовых сценариев. На данном этапе работы было реализовано 3 тестовых сценария для проверки неправильного определения геолокации со стороны

посещаемых электронных ресурсов, а также деградации скорости приёма и передачи данных.

Программный кода теста, проверяющего неправильное определение геолокации на информационных ресурсах компании Google:

```
def google_redirect(server_domains):
    command = r'curl -L http://www.google.com'
    ssh_output = ssh.run_command(command, server_domains)

    result = []

    for host in ssh_output:
        try:
            if host['output'] != -1:
                try:
                    match = re.search("(kGL:')([a-zA-Z]{2})(['])",
host['output'])
                    kgl = match.group(2).casefold()
                except AttributeError:
                    print(host['name'], 'Google response page code has no kGL
variable. kGL = N/A')
                    kgl = 'N/A'

                try:
                    country_code = host['name'].split('-')[2].casefold()

                    ...
                except IndexError:
                    ...

                if kgl == country_code or kgl == 'zz':
                    value = 0
                else:
                    value = 1

                result.append(dict(server_name=host['name'],
server_ip=host['ip'], country_code=country_code, kgl=kgl, value=value))
            else:
                result.append(dict(server_name=host['name'],
server_ip=host['ip'], country_code='N/A', kgl='N/A', value=-1))

        except Exception as e:
            print(host['name'], e)
            result.append(dict(server_name=host['name'],
server_ip=host['ip'], country_code='N/A', kgl='N/A', value=-1))
            continue

    return result
```

Как упоминалось ранее, дальнейшее развитие фреймворка с точки зрения добавления новых тестовых сценариев возможно путём реализации логики только самого тест-кейса, без внесения дополнительных изменений в другие части программного кода. Единственное требование, предъявляемое к

реализации новых тестов: все тесты должны возвращать результат в стандартизированном виде для его дальнейшей обработки. В данном случае — в структуре данных типа словарь, в котором обязательно должны быть указаны данные сервера и определяющее значение теста, показывающее его успешность или ключевой показатель. С выполнением этого условия в словарь может быть включено неограниченное количество других данных о выполнении теста.

3.2.3 Параллельный запуск тестов на всех серверах

Тестирование происходит в параллельном режиме, что позволяет установить соединение, выполнить один тестовый сценарий, обработать и отправить результаты в систему Prometheus на более чем 1000 серверах менее чем за 4 минуты.

Параллелизация реализована с помощью стороннего Python-модуля parallel-ssh, который позволяет создавать неограниченное количество асинхронных параллельных SSH-соединений [22]. Программный код создания соединения с командами для выполнения на серверах и извлечения результатов из ответного сообщения:

```
from gevent import monkey
monkey.patch_all()

from pssh.utils import load_private_key  # noqa
from pssh.pssh_client import ParallelSSHClient  # noqa
from config import const  # noqa
import socket  # noqa

...

def create_parallel_ssh_client(server_domains):
    pkey = load_private_key(const.PKEY)
    client = ParallelSSHClient(server_domains, pkey=pkey, user='ci')

    return client


def run_command(command, server_domains):
    try:
        ssh_client = create_parallel_ssh_client(server_domains)

        if ssh_client is None:
            return
```

```
        run_command_output = ssh_client.run_command(command,
stop_on_errors=False)
        ssh_client.join(run_command_output)

        output = []

        for host, host_output in run_command_output.items():
            server = __get_server_info(host)

            if not host_output.exception and host_output.exit_code == 0:
                stdout = ''

                try:
                    for line in host_output.stdout:
                        stdout += line
                except UnicodeDecodeError:
                    pass

                if stdout:
                    output.append(dict(name=server['name'], ip=server['ip'],
output=stdout))
                else:
                    output.append(dict(name=server['name'], ip=server['ip'],
output=-1))

            else:
                print('Exit code: {}, Exception:
{}'.format(host_output.exit_code, host_output.exception))
                output.append(dict(name=server['name'], ip=server['ip'],
output=-1))

        return output
    except Exception as e:
        print(e)
```

3.2.4 Обработка результатов тестирования

Был создан модуль parser_lib.py, в котором могут быть реализованы различные способы обработки и экспорта результатов данных. На данном этапе работы был реализован экспорт данных в систему Prometheus.

Prometheus — это инструментарий с открытым исходным кодом для системного мониторинга и оповещения, разработанный в 2012 году. Предоставляет возможности сбора и хранения как системных, так и пользовательских данных [23].

Чтобы реализовать мониторинг данных посредством Prometheus, необходимо интегрировать в свой программный код одну из клиентских библиотек Prometheus. Таким образом, была использована клиентская

библиотека prometheus_client для языка Python, с помощью которой выполняется экспорт данных в систему Prometheus [24].

Программный код подготовки и экспорта результатов тестирования в Prometheus:

```
from config import const
import prometheus_client as prom_client


def prom_parse_test_output(test_type, test_output, node_exporter=True,
pushgateway=True):
    try:
        test_name = 'node_test_' + test_type
        registry = prom_client.CollectorRegistry()
        gauge = prom_client.Gauge(test_name, '',
labelnames=list(test_output[0].keys()), registry=registry)

        for test_dict in test_output:
            value = test_dict.pop('value')
            gauge.labels(*list(test_dict.values())).set(value)
            test_dict['value'] = value  # because dict is mutable

        # Parse test output without TYPE and HELP noise by Prometheus
        output = []
        for metric in registry.collect():
            for name, labels, value in metric.samples:
                if labels:
                    labelstr = '{{{0}}}'.format(','.join(
                        ['{0}="{1}"'.format(
                            k, v.replace('\\', r'\\').replace('\n',
r'\n').replace('"', r'\"'))
                            for k, v in sorted(labels.items())]))
                else:
                    labelstr = ''
                output.append('{0}{1} {2}\n'.format(name, labelstr,
repr(float(value))))

        metrics = ''.join(output).encode('utf-8')

        if node_exporter:
            ...

        if pushgateway:
            prom_client.pushadd_to_gateway(const.PROM_PUSHGATEWAY_URL,
job=test_name, registry=registry)
            print('Test output was pushed to gateway
{}:\n{}'.format(const.PROM_PUSHGATEWAY_URL, metrics.decode('utf-8')))
    except KeyError:
        print("Wrong test results. Note that the dictionary key ['value'] is
required")
        raise
    except Exception as e:
        print(e)
...
```

3.2.5 Непрерывная интеграция

Расписание в CircleCI настроено таким образом, что тесты запускаются раз в сутки. На рисунке 10 представлена часть списка автоматизированных сборок фреймворка и последующего выполнения тестов.

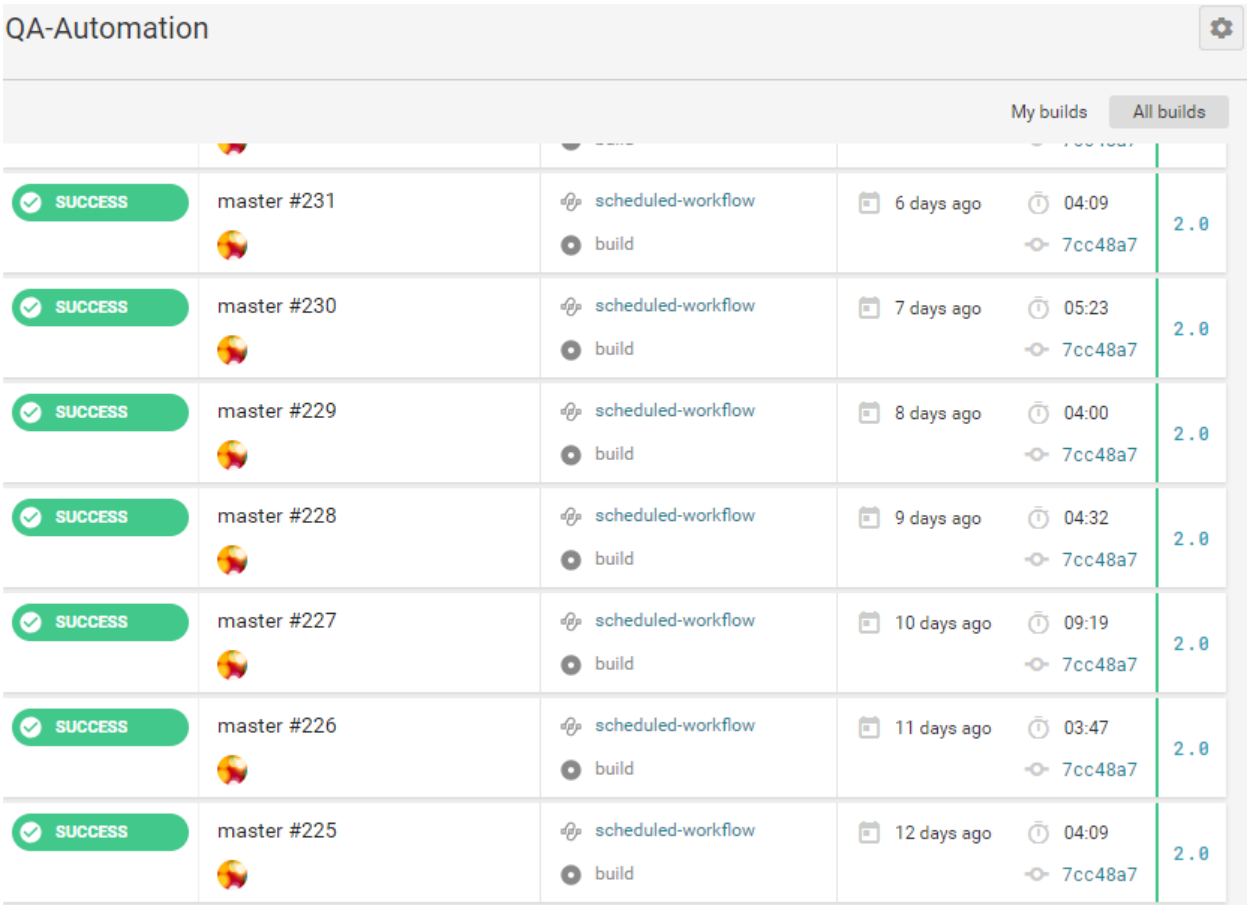

Рисунок 10. Список сборок фреймворка в CircleCI

На рисунке 11 представлены этапы сборки, куда входит установка ключа доступа к серверам, установка необходимых зависимостей и запуск тестов.

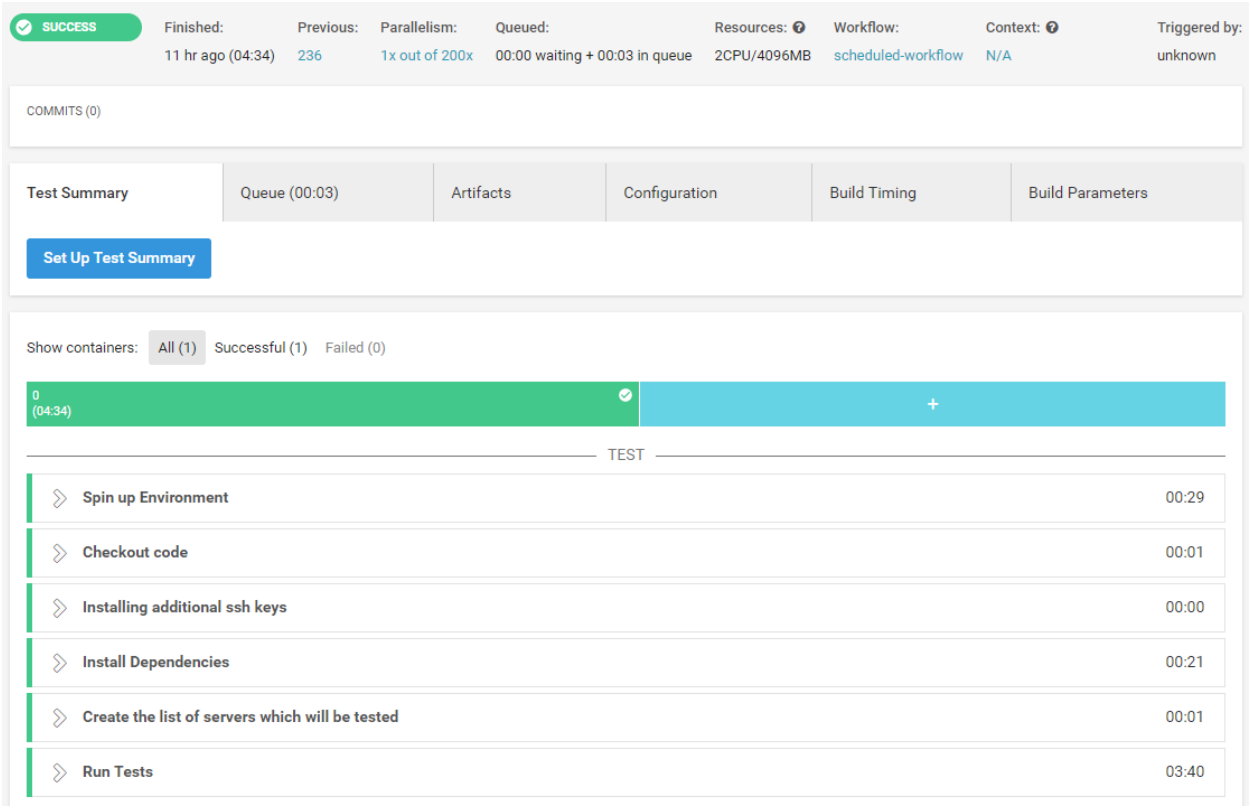

Рисунок 11. Этапы сборки в CircleCI

3.2.6 Страница с результатами тестов в Grafana

Результаты тестирования были представлены в удобном для мониторинга виде с помощью системы визуализации данных Grafana. В Grafana поддерживается интеграция со многими базами данных, системами сбора информации, в частности с Prometheus [25].

Для визуализации результатов тестирования в Grafana была создана отдельная страница, на которой данные представлены в виде таблиц, графиков и диаграмм. Кроме того, показан процент отрицательных тестов с разделением по странам, в которых располагаются удалённые сервера, а также распределение серверов на карте мира. Статистическая страница представлена на рисунке 9.

Необходимые данные из системы Prometheus были получены и представлены в соответствующем виде с помощью написания структурированных запросов на языке PromQL, разработанном Prometheus для выполнения этой задачи.

Пример запроса на языке PromQL:

```
(count(node_test_googleredirect{country_code='ru'} == 1) OR vector(0))
    / (count(node_test_googleredirect{country_code='ru'}) OR vector(1)) * 100
```

Рисунок 12. Статистическая страница в Grafana

3.3 Реализация тестирования пользовательского интерфейса SDK-приложения

Тестирование пользовательского интерфейса заключается в программной имитации действий потенциального пользователя приложения.

При этом подобные тесты являются наиболее нестабильными, потому что в случае даже незначительного изменения в графическом интерфейсе может возникнуть ситуация, когда нужно изменять логику большого количества тестов. Особенно этому подвержены те тесты, которые записаны с помощью специального рекордера, который запоминает действия пользователя и генерирует программный код для их повторения.

Многие Android-приложения разработаны с использованием шаблона проектирования Model-View-Presenter (Модель-Вид-Представитель), который предназначен для облегчения автоматизированного модульного тестирования и улучшения разделения ответственности в презентационной логике (отделения логики от отображения). На рисунке 13 показана ситуация, когда после внесения изменения в пользовательский интерфейс все связанные с изменённым элементом тесты перестали выполняться. В этом случае QA-специалист по автоматизированному тестированию должен переработать программную реализацию каждого из этих тестов по отдельности, чтобы вернуть их в рабочее состояние. На это нужно затратить много времени. К тому же, подобная ситуация в будущем может повториться вновь, что делает автоматизацию тестирования абсолютно неэффективным и не оправдывающим себя процессом.

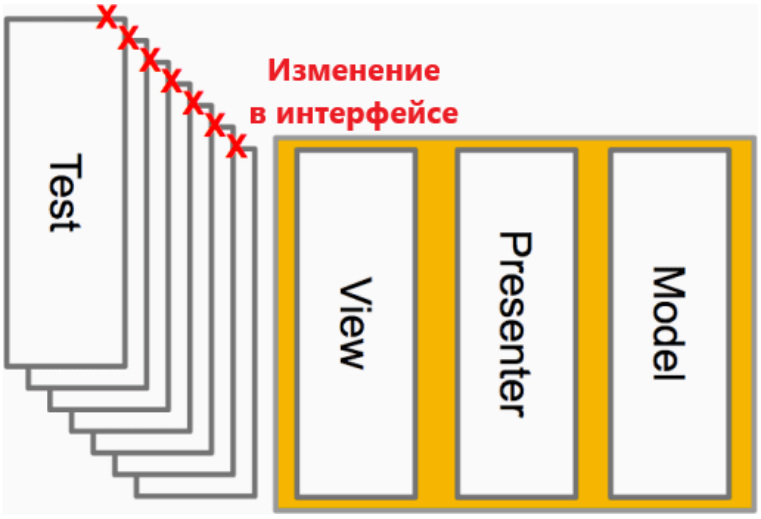

Рисунок 13. Состояние тестов после внесения изменений в пользовательский интерфейс

Чтобы избежать такой ситуации, важно выбрать и применить правильный подхода к реализации тестов с точки зрения архитектуры тестового пакета или фреймворка.

3.3.1 Паттерн Testing Robots как аналог Page Object

В настоящий момент в области автоматизации тестирования существует один наиболее известный шаблон проектирования — Page Object [18].

Page Object — это шаблон проектирования, который широко используется в автоматизированном тестировании и позволяет разделять логику выполнения тестов от их реализации. Page Object моделирует страницы тестируемого приложения в качестве объектов в коде. В результате его использования получаются отдельные классы, отвечающие за работу с HTML каждой конкретной веб-страницы. Такой подход значительно уменьшает объем повторяющегося кода, потому что одни и те же объекты страниц можно использовать в различных тестах. Основное преимущество Page Object заключается в том, что в случае изменения пользовательского интерфейса, можно выполнить исправление только в одном месте, а не исправлять каждый тест, в котором этот интерфейс используется [18].

Однако паттерн Page Object применяется в первую очередь для тестирования веб-сайтов. Учитывая, что работа с веб-страницей очень схожа с работой со страницей (Activity) в мобильном приложении, существует паттерн Testing Robots, который является аналогом Page Object, но применяется для тестирования мобильных приложений, в частности написанных для платформы Android.

Паттерн Testing Robots был предложен разработчиком компании Google, Джейком Вортоном [26], который является создателем множества модульных библиотек и инструментов для разработки ПО. Он подразумевает реализацию специального класса-робота (Robot), в котором описаны все элементы, действия и проверки ожидаемого поведения отдельно взятой

страницы приложения. При таком подходе, в случае внесения изменения в пользовательский интерфейс, QA-специалисту нужно будет модифицировать логику работы с изменённым элементов страницы только в одном месте — в классе-роботе (рис. 14), что значительно упрощает поддержку тестов.

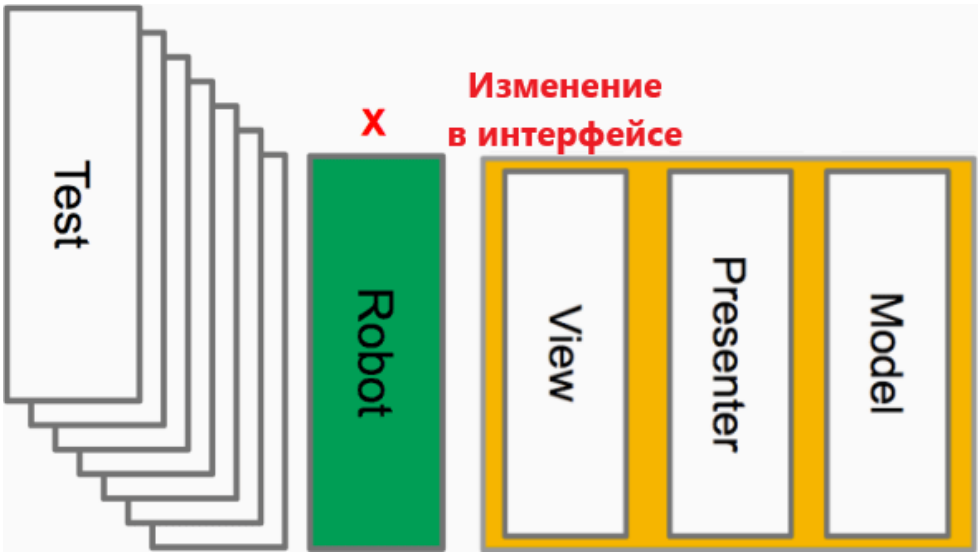

Рисунок 14. Состояние класса-робота после внесения изменений в пользовательский интерфейс

Автоматизация тестирования пользовательского интерфейса мобильного SDK-приложения на платформе Android была выполнена именно с использованием паттерна Testing Robots.

3.3.2 Реализация класса MainRobot

Как уже было отмечено ранее, для реализации тестов был выбран фреймворк Espresso. В ходе реализации был выявлен один несущественный, но заметный недостаток, который влияет на процесс написания программного кода тестов.

Синтаксис Espresso разработан таким образом, что для того, чтобы выполнить какое-либо действие с элементом на странице, необходимо запрограммировать сразу несколько шагов. Из-за этого программный код становится перегруженным и трудночитаемым. Пример одного пользовательского действия, реализованного с помощью Espresso:

```
onView(withText(R.string.toast_text))
    .inRoot(withDecorView(not(mActivityRule
        .getActivity()
        .getWindow()
        .getDecorView())))
    .check(matches(isDisplayed()));
```

Чтобы упростить написание тестов и сделать их программный код более читаемым, был использован фреймворк Barista, который является обёрткой вокруг Espresso. Barista делает разработку UI-теста более быстрой, легкой и предсказуемой, предоставляя простой API-интерфейс к функциональности Espresso [27].

Для того, чтобы показать разницу реализации пользовательских действий с помощью Espresso и Barista, далее приведён код одного и того же действия, запрограммированного с помощью данных фреймворков соответственно.

- Нажатие левой кнопкой мыши по элементу:

```
onView(withId(R.id.button)).perform(click());  // Espresso

clickOn(R.id.button);  // Barista
```

- Проверка наличия указанного текста в указанном текстовом поле:

```
onView(withId(R.id.text_view)).check(matches(withText("text")));  // Espresso

assertContains(R.id.text_view, "text");  // Barista
```

Таким образом, класс-робот был реализован именно с использованием фреймворка Barista. Часть программного кода на языке Java, который представляет класс MainRobot, описывающего страницу MainActivity тестируемого приложения:

```
public class MainRobot {
    private final long waitTimeout = TimeUnit.SECONDS.toMillis(15);

    @NonNull
    public MainRobot backendUrl(@NonNull final String url) {
        clickOn(R.id.backend_url);
        writeTo(R.id.backend_url, url);
        closeSoftKeyboard();
        assertContains(R.id.backend_url, url);
        closeAutoCompleteSuggestions();
        return this;
    }
```

```
    @NonNull
    public MainRobot authType(@NonNull final String type) {
        clickOn(R.id.auth_type);
        writeTo(R.id.auth_type, type);
        closeSoftKeyboard();
        assertContains(R.id.auth_type, type);
        return this;
    }

    ...

    @NonNull
    public MainRobot login() {
        clickOn(R.id.login_btn);
        return this;
    }

    @NonNull
    public MainRobot loginAs(@NonNull final Login login) {
        backendUrl(login.backendUrl);
        authType(login.authType);
        authToken(login.authToken);
        carrierId(login.carrierId);
        return login();
    }

    ...

    @NonNull
    public MainRobot assertLoggedIn() {
        scrollTo(R.id.logged_in_info);
        assertContains(R.id.logged_in_info, "Logged in");
        return this;
    }

    ...

    @NonNull
    public MainRobot assertConnected() {
        scrollTo(R.id.vpn_status);
        onView(isRoot()).perform(waitIdWithText(R.id.vpn_status, "CONNECTED",
waitTimeout));
        assertContains(R.id.vpn_status, "CONNECTED");
        return this;
    }

    @NonNull
    public MainRobot assertToastIsDisplayed(@NonNull final String toast) {
        onView(withText(toast)).inRoot(new
ToastMatcher()).check(matches(isDisplayed()));
        return this;
    }

    ...
}
```

3.3.3 Реализация тестов

Тесты были реализованы с использованием фреймворка JUnit, который помогает разделить тесты на отдельные методы и отвечает за их изолированный запуск и выполнение [28].

В результате применения паттерна Testing Robots, каждый тест представляет собой последовательность шагов, описанных в классе-роботе. Программный код тестов не нагромождён взаимодействием с пользовательским интерфейсом и максимально понятно описывает действия, выполняемые в тест-кейсе. Часть программного кода:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class MainActivityTest {
    @Rule
    @NonNull public final ActivityTestRule<MainActivity> atr = new
ActivityTestRule<>(MainActivity.class);
    @NonNull private final MainRobot mainRobot = new MainRobot();
    ...

    @Before
    public void setUp() throws UiObjectNotFoundException {
        mainRobot = new MainRobot();
        enableWifi(true, atr);
    }

    @Test
    public void appName_onResume_shouldBeDisplayed() {
        mainRobot.assertAppNameIsDisplayed();
    }

    ...

    @Test
    public void login_whenLoggedOut_shouldLogin() {
        mainRobot
                .loginAs(login)
                .assertLoggedIn();
    }

    @Test
    public void userId_afterLogin_shouldBeDisplayed() {
        mainRobot
                .loginAs(login)
                .assertLoggedIn()
                .assertUserIdIsDisplayed();
    }

    @Test
    public void logout_whenLoggedIn_shouldLogout() {
        mainRobot
                .loginAs(login)
                .assertLoggedIn()
```

```
                .logout()
                .assertLoggedOut();
    }

    @Test
    public void creds_whenLoggedIn_shouldGetCreds() {
        mainRobot
                .loginAs(login)
                .assertLoggedIn()
                .creds()
                .assertCredsAreValid();
    }

    @Test
    public void connect_whenLoggedInWithCreds_shouldConnect() {
        mainRobot
                .loginAs(login)
                .assertLoggedIn()
                .creds()
                .assertCredsAreValid()
                .connect()
                .assertConnected();
    }

    ...
}
```

3.3.4 Запуск тестов в Firebase Test Lab

На разных Android-устройствах приложение может вести себя по-разному. Поэтому имеет смысл запускать тесты на большом количестве устройств, отличающихся по модели и производителю, а также на разных версиях операционной системы.

Организация подобного парка устройств может потребовать больших финансовых вложений для компании-разработчика. Однако, существуют сторонние сервисы, предоставляющие определённый набор устройств, на которые можно установить тестируемое приложение и запустить тесты. Одним из таких сервисов является Firebase Test Lab от компании Google.

Firebase Test Lab предоставляет облачную инфраструктуру для тестирования Android-приложений на широком спектре устройств и их конфигураций. Результаты тестов — включая логи, видео и скриншоты — доступны в пользовательском проекте в консоли Firebase Test Lab. Запуск тестов возможен с веб-сайта сервиса, из Android Studio или из командной строки [29].

В рамках работы над магистерской диссертацией был выполнен запуск тестов в Firebase Test Lab (рис. 15) на нескольких устройствах из среды разработки Android Studio.

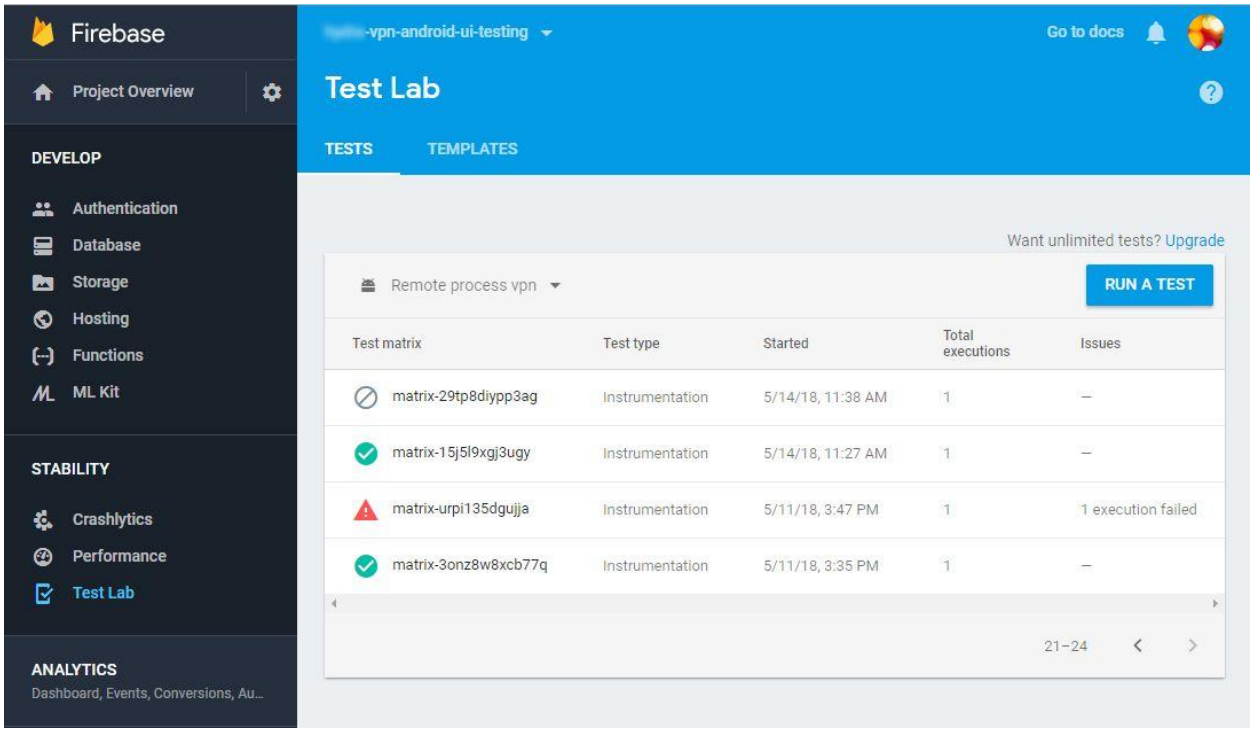

Рисунок 15. Запуск тестов в Firebase Test Lab

На рисунке 16 представлены результаты выполнения нескольких тестов на одном устройстве (в IDE Android Studio). Зелёным помечены тесты, завершившиеся успешно, красным — не соответствующие ожидаемому поведению программы. На основе провалившихся тестов разработчику может быть сообщено о существующем баге.

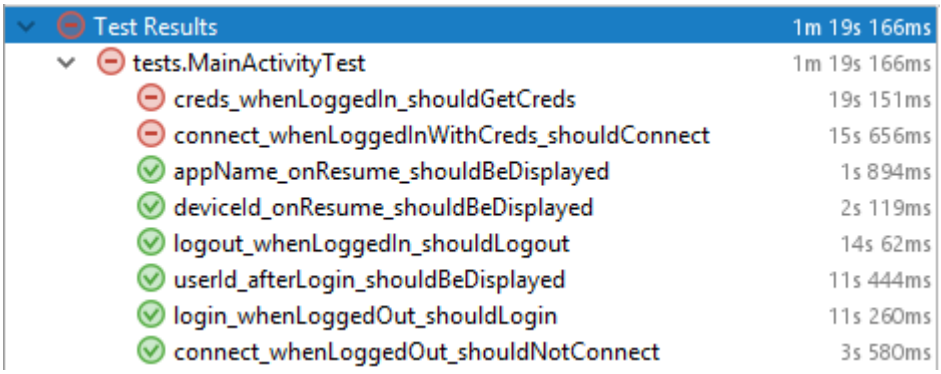

Рисунок 16. Результаты выполнения тестов

Сервис Firebase Test Lab предоставляет 5 бесплатных тестовых запусков в день. Благодаря проведённому исследованию и пробным запускам, в ближайшее время в компании планируется приобретение возможности дополнительных запусков тестов на облачных устройствах в данном сервисе.

3.4 Результаты

В результате были реализованы 2 решения для автоматизации тестирования двух разных частей одного VPN-сервиса.

Для тестирования удалённых серверов VPN-сервиса был разработан фреймворк на языке программирования Python, объединяющий в себе параметризацию запуска тестов, реализацию тестовых сценариев, обработку и экспорт результатов тестирования в систему сбора данных Prometheus. Для визуализации и мониторинга результатов тестирования по одному из тестовых сценариев была разработана статистическая страница в системе Grafana. Для сборки и запуска тестов по расписанию была выполнена интеграция с системой непрерывной интеграции CircleCI. Для решения заявленных проблем было реализовано 3 тестовых сценария.

Архитектура фреймворка позволяет добавлять новые тестовые сценарии, не изменяя при этом другие части программного кода фреймворка. Текущая реализация удовлетворяет настоящие потребности организации, однако позволяет выполнять относительно простые тесты, которые могут быть выполнены с использованием штатных средств тестируемого сервера и не требуют установки дополнительных зависимостей на него. В будущем при возникновении необходимости реализации более сложных тестов, требующих, например, автоматизации с помощью Selenium, одним из возможных решений может быть помещение тестового фреймворка с необходимыми зависимостями в виртуальный образ и его последующий запуск на удалённых серверах с помощью Docker.

Для тестирования пользовательского интерфейса SDK-приложения были реализованы автоматизированные тесты на языке программирования

Java. При реализации были использованы такие инструменты, как Espresso, UI Automator, Barista, JUnit, паттерн проектирования Testing Robots и сервис Firebase Test Lab. Всего на данном этапе работы было реализовано 30 существующих тест-кейсов регрессионного тестирования.

Реализованные тестовые решения внедрены и используются в процессах разработки и поддержки VPN-сервиса и сопутствующего программного обеспечения в компании ООО «Сибэдж».

Автоматизация позволила повысить эффективность процесса тестирования и качество программного продукта. Это стало возможным благодаря тому, что ошибки, возникающие в процессе работы удалённых серверов VPN-сервиса, теперь обнаруживаются на раннем этапе, не доходя до конечного пользователя.

Также снизились ресурсозатраты и повысилась скорость выполнения процесса тестирования за счёт автоматизации регрессионного тестирования SDK-приложения, на которое ранее специалисты по ручному тестированию тратили значительное количество времени с частой периодичностью.

4 ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ

4.1 Введение

Выполнение работы в рамках данной магистерской диссертации заключается в реализации автоматизированного тестирования и мониторинга удалённых серверов VPN-сервиса. Это включает в себя разработку и настройку программного обеспечения для автоматизации процесса тестирования и его последующую непрерывную интеграцию.

В настоящее время перспективность разработки программного обеспечения определяется такими факторами, как наличие потенциальных потребителей конечного продукта, оценки существующих аналогов и конкурентных решений, анализ качества разработки и её перспективности на рынке.

Таким образом, целью раздела «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» является проектирование и создание конкурентоспособных разработок, технологий, отвечающих современным требованиям в области ресурсоэффективности и ресурсосбережения.

Достижение цели обеспечивается решением задач:

- оценка коммерческого потенциала и перспективности проведения научных исследований;
- определение возможных альтернатив проведения научных исследований, отвечающих современным требованиям в области ресурсоэффективности и ресурсосбережения;
- планирование научно-исследовательских работ;
- определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования.

4.2 Предпроектный анализ

4.2.1 Потенциальные потребители результатов проекта

Целевым рынком данной разработки являются организации, занимающиеся разработкой программного обеспечения и предоставляющие отдельные узконаправленные сервисы широкому кругу пользователей посредством определённой настройки сети распределённых удалённых серверов.

Для того, чтобы определить для каких организаций необходима данная разработка, необходимо провести сегментирование целевого рынка. Сегментацию можно произвести по следующим двум критериям: размер организации и вид тестирования программного обеспечения или работающего терминала (сервера). Карта сегментирования представлена в таблице 1.

Таблица 1. Карта сегментирования области тестирования ПО

		Вид тестирования программного обеспечения			
		Автоматизированное	Ручное	Аутсорсинг	Мониторинг и непрерывная интеграция
Размер организации	Крупные				
	Средние				
	Мелкие				

ООО «Сибэдж» (крупная)	Организация B (средняя)	Организация C (мелкая)

Малым или молодым компаниям-разработчикам ПО с экономической точки зрения не выгодно иметь собственный отдел тестирования, так как разрабатываемые и поддерживаемые ими проекты и системы уступают в сложности и масштабе проектам более крупных организаций. Поэтому такие компании либо вовсе не используют тестирование, жертвуя при этом качеством разработки, либо отдают этот процесс на аутсорсинг крупным организациям.

Средним компаниям тестирование необходимо, при этом сложность проектов делает необходимым применение только ручного тестирования, которое может носить непостоянный характер, или же не претендовать на максимальную точность при проверке программного обеспечения. При редкой необходимости автоматизации процесса тестирования, связанной со спецификой отдельно взятого проекта или заказчика, данная задача может быть передана на аутсорсинг в более крупную компанию.

Крупные разработчики программного обеспечения, как правило, занимаются наиболее сложными проектами, что выражается в больших длительности и ресурсозатратах. Поэтому для того, чтобы разработанное программное обеспечение было качественным, или же для обеспечения и мониторинга непрерывного функционирования сервиса или системы, необходимо применение всех видов тестирования. Кроме того, комбинация автоматизированного и ручного тестирования дополняется процессами непрерывной интеграции и мониторинга результатов тестирования, которые реализуется посредствами сторонних систем, используемых при разработке ПО.

Таким образом, разработка в рамках магистерской диссертации, заключающаяся в реализации автоматизированного тестирования и мониторинга, подходит для организаций типа «С». Компания ООО «Сибэдж», для нужд которой и осуществлялась разработка, относится именно к сегменту крупных компаний.

4.2.2 Анализ конкурентных технических решений

Одна из поставленных задач была выполнена посредством программной реализации собственного фреймворка. Тем не менее, существуют готовые решения, направленные на имитацию действий пользователя, позволяющие автоматизировать различные административные задачи управления и настройки распределённых систем [30]. Подобные решения могут быть полезны при автоматизации простых задач, в частности задач тестирования отдельной системы на локальном уровне. Реализация поставленной задачи с помощью таких решений потребует в несколько раз больше времени и ресурсов, будет трудноподдерживаемой с точки зрения вхождения в проект новых специалистов и реализации дополнительных тестовых сценариев в будущем.

Для подтверждения приоритетности реализации собственного решения была составлена карта сравнения конкурентных систем. В качестве конкурентных систем были рассмотрены такие решения, как IBM Rational Functional Tester ($Б_{к1}$) [31] и AutoIT ($Б_{к2}$) [32]. Эти программные средства используются для создания скриптов автоматизации для различных платформ. Такие скрипты полезны для выполнения часто повторяющихся задач на большом количестве компьютеров. Оценочная карта сравнения представлена в таблице 2.

Таблица 2. Оценочная карта сравнения конкурентных систем

Критерии оценки	**Вес крите-рия**	**Баллы**			**Конкуренто-способность**		
		$Б_ф$	**$Б_{к1}$**	**$Б_{к2}$**	**$К_ф$**	**$К_{к1}$**	**$К_{к2}$**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Технические критерии оценки ресурсоэффективности							
1. Повышение производительности	0,1	5	3	3	0,5	0,3	0,3
2. Удобство в эксплуатации	0,1	5	1	1	0,5	0,1	0,1
3. Надежность	0,11	4	5	4	0,44	0,55	0,44
4. Безопасность	0,12	5	5	5	0,6	0,6	0,6
5. Потребность в ресурсах памяти	0,02	5	3	4	0,1	0,06	0,08
6. Функциональная мощность	0,07	4	3	2	0,28	0,21	0,14

Критерии оценки	Вес крите-рия	Баллы			Конкуренто-способность		
		$Б_ф$	$Б_{к1}$	$Б_{к2}$	$К_ф$	$К_{к1}$	$К_{к2}$
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
7. Простота эксплуатации	0,08	4	3	2	0,32	0,24	0,16
8. Качество интеллектуального интерфейса	0,06	4	5	3	0,36	0,3	0,18
Экономические критерии оценки эффективности							
1. Конкурентоспособность продукта	0,05	5	5	5	0,25	0,25	0,25
2. Уровень проникновения на рынок	0,03	2	4	4	0,06	0,12	0,12
3. Цена	0,2	5	3	5	1	0,6	1
4. Послепродажное обслуживание	0,03	5	3	3	0,15	0,09	0,09
5. Срок выхода на рынок	0,01	4	4	4	0,04	0,04	0,04
6. Наличие сертификации разработки	0,02	5	5	5	0,1	0,1	0,1
Итого	**1**				**4,7**	**3,46**	**3,6**

Из приведённого сравнения собственного разработанного фреймворка с программными средствами IBM Rational Functional Tester и AutoIT видно, что уязвимость конкурентных технических решений связана прежде всего с их неудобством эксплуатации и низкой производительностью труда с точки зрения реализации задач тестирования целевой системы (удалённых серверов VPN-сервиса).

Таким образом, выбранный подход реализации собственного фреймворка является более предпочтительным, чем использование существующих решений автоматизации, так как конечное решение включает в себя интеграцию нескольких сторонних систем. Такое взаимодействие реализуется непосредственно в программном коде, а настройка отдельных компонентов — в специальных конфигурационных файлах. Кроме того, при таком подходе добавление новых тестовых сценариев значительно упрощается, так как все подготовительные и вспомогательные этапы уже были реализованы единожды и не нуждаются в повторной реализации. Также в большинстве случаев отсутствует пользовательский интерфейс, что является весомым аргументом в пользу реализации собственного фреймворка.

4.2.3 QuaD-анализ

Для описания качества новой разработки и её перспективности на рынке была использована технология QuaD. Оценка разработки с учетом её технических и экономических особенностей, создания и коммерциализации представлена в таблице 3.

Таблица 3. QuaD-анализ

Критерии оценки	**Вес критерия**	**Баллы**	**Максимальный балл**	**Относительное значение (3/4)**	**Средневзвешенное значение (5x2)**
1	**2**	**3**	**4**	**5**	**6**
Показатели оценки качества разработки					
1. Повышение производительности труда пользователя	0,1	95	100	0,95	0,095
2. Удобство в эксплуатации (соответствует требованиям потребителей)	0,1	95	100	0,95	0,095
3. Надёжность	0,11	80	100	0,80	0,088
4. Безопасность	0,12	90	100	0,90	0,108
5. Потребность в ресурсах памяти	0,02	90	100	0,90	0,018
6. Функциональная мощность (предоставляемые возможности)	0,07	80	100	0,80	0,056
7. Простота эксплуатации	0,08	80	100	0,80	0,064
8. Качество интеллектуального интерфейса	0,06	80	100	0,80	0,048
Показатели оценки коммерческого потенциала разработки					
9. Конкурентоспособность продукта	0,05	95	100	0,95	0,0475
10. Уровень проникновения на рынок	0,03	20	100	0,20	0,006
11. Цена	0,2	95	100	0,95	0,19
12. Послепродажное обслуживание	0,03	80	100	0,80	0,024
13. Срок выхода на рынок	0,01	70	100	0,70	0,007
14. Наличие сертификации разработки	0,02	100	100	1	0,02
Итого	**1**				**86,65%**

В результате проведенного анализа перспективность разработки равна 86,65%. Поскольку значение показателя входит в промежуток от 80% до 100%, данная разработка является перспективной.

4.2.4 Диаграмма Исикавы

Одной из проблематик предприятия, для решения которой выполнялась данная работа, является отсутствие автоматизированного тестирования и мониторинга удалённый серверов VPN-сервиса. Для описания причин проблемы была использована диаграмма Исикавы, в которой показываются причинно-следственные связи. Диаграмма представлена на рисунке 17.

Рисунок 17. Диаграмма Исикавы

Диаграмма Исикавы — графический способ исследования и определения наиболее существенных причинно-следственных взаимосвязей между факторами и последствиями в исследуемой ситуации или проблеме.

4.2.5 SWOT-анализ

Для комплексного анализа и исследования внешней и внутренней среды проекта применяется SWOT-анализ. Итоговая матрица SWOT с сильными и слабыми сторонами проекта реализации фреймворка, возможностями и угрозами представлена в таблице 4.

Таблица 4. Итоговая матрица SWOT

	Сильные стороны проекта: С1. Централизованное тестирование всех удалённых серверов сети. С2. Снижение временных затрат на тестирование. С3. Расширяемый функционал фреймворка. С4. Мониторинг результатов тестирования. С5. Повышение качества сервиса.	**Слабые стороны проекта:** Сл1. Необходима поддержка со стороны DevOps-специалиста. Сл2. Для расширения функционала необходима работа разработчика или специалиста по автоматизированному тестированию. Сл3. Отсутствие графического пользовательского интерфейса. Сл4. Невозможно выполнять тесты, требующие установки дополнительного ПО на сервер.
Возможности: В1. Добавление новых тестовых сценариев. В2. Добавление новых способов обработки результатов. В3. Параметризация списка серверов для тестирования. В4. Изменение способа отображения информации. В5. Изменение расписания тестирования.	**Направления развития:** 1. В1С1С2С3С5 — добавление новых сценариев позволит увеличить тестовое покрытие и повысить качество сервиса. 2. В2В4С4 — работа со способами мониторинга позволит улучшить отображение и информативность результатов тестирования.	**Сдерживающие факторы:** 1. В1Сл4 — для реализации более сложных тестов необходима существенная модернизация или контейнеризация решения. 2. В3В5Сл1Сл3 — настройка плана тестирования может стать проблемой для неопытных специалистов. 3. В1В2Сл2 — необходимость привлечения

	3. В3В5С2С5 — управление расписанием и списком серверов позволит проводить точечные проверки.	специалистов по разработке может замедлить развитие решения.
Угрозы: У1. Изменение принципов работы тестируемых с сервера ресурсов. У2. Сбои в работе системы непрерывной интеграции. У3. Изменение работы используемых протоколов VPN. У4. Несвоевременное предоставление доступа к новым серверам со стороны DevOps-специалистов.	**Угрозы развития:** 1. У1У2У3С1С4 — изменение принципов работы задействованных систем может нарушить процесс тестирования и мониторинга. 2. У4С4С5 — отсутствие доступа на сервера может стать причиной необнаружения существующих ошибок.	**Уязвимости:** 1. У1У3Сл2Сл4 — необходимость вмешательства разработчика может повлечь отказ от использования решения и возврат к старым методам тестирования, требующим меньшую квалификацию исполнителей. 2. У2У4Сл1Сл3 — сложности в настройке могут существенно замедлить процесс расширения тестирования, что повлечёт за собой отсутствие актуальной информации об удалённых серверах VPN-сервиса.

Из итоговой матрицы SWOT видно, что несмотря на наличие таких сдерживающих факторов и уязвимостей, как необходимость вмешательства специалистов по разработке ПО или DevOps для настройки или расширения функционала решения (поля пересечений: В1В2Сл2 и У1У3Сл2Сл4), возможности, которые предоставляет разработка, автоматизация и сокращение временных затрат на процесс тестирования (поля пересечений: В1С1С2С3С5, В2В4С4, В3В5С2С5) позволят повысить приоритет работы над данным решением для данных специалистов, что в свою очередь понизит вероятность возникновения описанных рисков и уязвимостей.

4.3 Инициация проекта

В рамках процессов инициации определяются изначальные цели и содержание и фиксируются изначальные финансовые ресурсы. Определяются

внутренние и внешние заинтересованные стороны проекта, которые будут взаимодействовать и влиять на общий результат научного проекта.

4.3.1 Цели и результат проекта

Перед определением целей необходимо перечислить заинтересованные стороны проекта. Информация по заинтересованным сторонам представлена в таблице 5.

Таблица 5. Заинтересованные стороны проекта

Заинтересованные стороны проекта	**Ожидание заинтересованных сторон**
Организация-заказчик	Автоматизация тестирования и мониторинга
Пользователь	Снижение временных затрат на тестирование, упрощение процесса
Разработчик	Прибыль
Научный руководитель, студент	Готовая магистерская диссертация

Цели и результат проекта в части выполнения одной из двух основных задач магистерской диссертации, а именно реализации фреймворка для автоматизации тестирования и мониторинга удалённых серверов VPN-сервиса, представлены в таблице 6.

Таблица 6. Цели и результат проекта

Цели проекта	• Спроектировать решение для автоматизации процесса тестирования удалённых серверов. • Реализовать фреймворк для тестирования с возможностью его дальнейшего расширения путём добавления новых тестовых сценариев. • Реализовать интеграцию с системами сбора и визуализации данных. • Реализовать запуск тестов по расписанию с помощью системы непрерывной интеграции.
Ожидаемый результат проекта	Успешно внедрённая разработка внутри предприятия-заказчика
Критерии приёмки результата проекта	Успешно автоматизированный процесс тестирования и мониторинга с помощью реализованного фреймворка и заданных тестовых сценариев
Требования к результату проекта	• Выполнены все пункты технического задания. • Разработанный функционал полностью соответствует проектным решениям.

4.3.2 Организационная структура проекта

Организационная структура проекта представлена в таблице 7.

Таблица 7. Организационная структура проекта

№	ФИО, основное место работы, должность	Роль в проекте	Функции
1	Назимок Павел Павлович, ООО «Сибэдж», тестировщик	Разработчик	• Проектирование. • Реализация. • Внедрение.
2	Шилов Василий Андреевич, ООО «Сибэдж», руководитель отдела обеспечения качества	Руководитель проекта от предприятия	• Проверка разработки. • Помощь во внедрении.
3	Шестаков Николай Александрович, ТПУ, к.т.н., доцент отделения ИТ	Научный руководитель	• Составление научных целей и задач. • Проверка документации.

4.4 Определение возможных альтернатив разработки

Для определения альтернативных вариантов реализации технической задачи используется морфологический подход. Морфологическая матрица для составляющих реализации рассматриваемого проекта представлена в таблице 8.

Таблица 8. Морфологическая матрица

	1	2	3
А. Язык разработки	Python	Java	Ruby
Б. Среда разработки	PyCharm	IntelliJ IDEA	RubyMine
В. Система мониторинга и данных и оповещения	Prometheus	InfluxDB	Elasticsearch
Г. Система визуализации данных	Grafana	Kibana	Zabbix
Д. Система непрерывной интеграции	CircleCI	TravisCI	Jenkins

Из полученной морфологической матрицы можно получить достаточно много вариантов, применяя любые комбинации пунктов В, Г, Д. При этом должно выполняться условие одновременного использования определённых вариантов пунктов А и Б, а именно: А1Б1, А2Б2, А3Б3.

Таким образом, из полученной морфологической матрицы можно получить как минимум 3 варианта реализации проекта:

- Исполнение 1: А1Б1В1Г1Д1;
- Исполнение 2: А2Б2В2Г2Д2;
- Исполнение 3: А3Б3В3Г3Д3.

В рамках магистерской диссертации реализуется первый вариант исполнения, а 2 других необходимы для проведения сравнительного анализа (в частности, в разделе 4.6, табл. 18).

4.5 Планирование научно-исследовательских работ

4.5.1 Ограничения и допущения

Ограничения проекта представлены в таблице 9.

Таблица 9. Ограничения проекта

Фактор	**Ограничения**
Бюджет проекта	140000 рублей
Источник финансирования	ООО «Сибэдж»
Сроки проекта	29.01.2018-04.06.2018
Фактическая дата утверждения плана управления проектом	29.01.2018
Плановая дата завершения проекта	04.06.2018

4.5.2 Структура работ в рамках проекта

В таблице 10 представлено распределение исполнителей по видам работ.

Таблица 10. Перечень этапов работ и распределение исполнителей

Основные этапы	**№**	**Содержание работ**	**Должность исполнителя**
Постановка задач	1	Описание требований	Руководитель от предприятия, научный руководитель
	2	Описание бизнес-процессов	Руководитель от предприятия
	3	Анализ предметной области	Руководитель от предприятия, научный руководитель, студент

Основные этапы	**№**	**Содержание работ**	**Должность исполнителя**
	4	Разработка технического задания	Руководитель от предприятия, студент
Проектирование	5	Проектирование интеграции с системами мониторинга и непрерывной интеграции	Руководитель от предприятия, студент
	6	Проектирование фреймворка	Руководитель от предприятия, студент
Разработка	7	Разработка параметризации тестовых параметров и списка серверов	Студент
	8	Разработка модуля обработки результатов тестирования	Студент
	9	Разработка тестовых сценариев	Студент
	10	Разработка модуля тестирования серверов	Студент
	11	Разработка модуля, отвечающего за параллельное соединение с серверами	Студент
	12	Разработка интеграции с системой мониторинга	Студент
Отладка решения и внедрение	13	Отладка обработки результатов и их отправки в систему мониторинга	Студент
	14	Отладка запуска сборок в системе непрерывной интеграции	Студент
	15	Отладка страницы отображения	Студент
	16	Внедрение разработки	Руководитель от предприятия, студент
Оформление документации	17	Написание пояснительной документации	Студент
	18	Проверка работы	Научный руководитель

4.5.3 Определение трудоёмкости выполнения работ

Для определения трудоёмкости проекта была построена диаграмма Ганта, которая предоставляет наглядное отображение графика и распределения работ между участниками проекта (табл. 11).

Таблица 11. Календарный план-график проведения работ

№	Содержание работ	Должность исполнителя	Продолжительность выполнения работ, 2018 год					
			Февраль	Март	Апрель	Май	Июнь	
1	Описание требований	РП, НР						
2	Описание бизнес-процессов	РП						
3	Анализ предметной области	РП, НР, С						
4	Разработка технического задания	РП, С						
5	Проектирование интеграции	РП, С						
6	Проектирование фреймворка	РП, С						
7	Разработка параметризации	С						
8	Разработка модуля обработки результатов	С						
9	Разработка тестовых сценариев	С						
10	Разработка модуля тестирования серверов	С						
11	Разработка модуля параллельного соединения	С						
12	Разработка интеграции с системой мониторинга	С						

№	Содержание работ	Должность исполнителя	Продолжительность выполнения работ, 2018 год																											
			Февраль						Март						Апрель						Май						Июнь			
13	Отладка обработки результатов и их отправки	С																												
14	Отладка запуска сборок	С																												
15	Отладка страницы отображения	С																												
16	Внедрение разработки	РП, С																												
17	Написание пояснительной документации	С																												
18	Проверка работы	НР																												

Студент (С) Научный руководитель (НР) Руководитель от предприятия (РП)

4.5.4 Бюджет научно-технического исследования

В состав бюджета входит стоимость всех расходов, необходимых для выполнения работ по магистерской диссертации. При формировании бюджета используется группировка затрат по следующим статьям:

- материальные затраты;
- основная заработная плата исполнителей темы;
- дополнительная заработная плата исполнителей темы;
- отчисления во внебюджетные фонды (страховые отчисления);
- накладные расходы.

4.5.4.1 Расчёт материальных затрат

Данная статья расходов включает стоимость всех материалов, используемых при разработке диссертации. При выполнении работы был использован один персональный компьютер в компании. Соответствующие материальные затраты представлены в таблице 12. Мелкие расходы (канцелярия, затраты на печать и пр.) могут быть отнесены к статье прочих расходов.

Таблица 12. Материальные затраты

Наименование	**Ед. измерения**	**Количество**	**Цена за ед., руб.**	**Сумма, руб.**
Персональный компьютер	шт.	1	25000	25000
			Итого по статье $C_м$	25000

4.5.4.2 Расчёт основной заработной платы исполнителей

В данную статью расходов включается основная заработная плата научного руководителя, руководителя от предприятия и студента. Расчёт выполняется на основе трудоемкости выполнения каждого этапа и величины месячного оклада исполнителя.

Трудоёмкость всех исполнителей в разные промежутки времени на протяжении выполнения магистерской диссертации была просуммирована и

представлена в виде затраченных дней. Таким образом, если согласно плану работа над проектом должна вестись в течение 127 дней, то реально затраченное время каждого исполнителя в днях отличается о данной цифры, а также от того, что можно увидеть из диаграммы Ганта (так как на ней декомпозиция происходит с точностью до дней, а не часов).

Расчет основной заработной платы приведен в таблице 13.

Таблица 13. Основная заработная плата исполнителей проекта

Исполнитель	**Оклад, руб./мес.**	**Средне-дневная ставка, руб.**	**Затраты времени, раб. дни**	**К.**	**Основная заработная плата, руб.**
Научный руководитель	26300	1374,5	2	1,3	3573,7
Руководитель от предприятия	50000	2511,2	4		13058,2
Студент	10000	502,2	90		58757,4
Итого по статье $З_{осн}$					75389,3

4.5.4.3 Расчёт дополнительной заработной платы исполнителей

В данную статью включается сумма выплат, предусмотренных законодательством о труде. Например, оплата очередных и дополнительных отпусков; оплата времени, связанного с выполнением государственных и общественных обязанностей; выплата вознаграждения за выслугу лет и т. п. (в среднем — 12% от суммы основной заработной платы).

Расчёты дополнительной заработной платы приведены в таблице 14.

Таблица 14. Дополнительная заработная плата исполнителей проекта

Исполнитель	**Основная заработная плата, руб.**	**К.**	**Дополнительная заработная плата, руб.**
Научный руководитель	3573,7	0,12	428,8
Руководитель от предприятия	13058,2		1567,0
Студент	58757,4		7050,9
Итого по статье $З_{доп}$			9046,7

4.5.4.4 Расчёт итоговой заработной платы исполнителей

Согласно расчётам, приведённым в таблицах 13 и 14, была посчитана итоговая заработная плата исполнителей, которая представлена в таблице 15.

Таблица 15. Заработная плата исполнителей

Исполнитель	**Основная зарплата, руб.**	**Дополнительная зарплата, руб.**	**Итоговая зарплата, руб.**
Научный руководитель	3573,7	428,8	4002,5
Руководитель от предприятия	13058,2	1567,0	14625,2
Студент	58757,4	7050,9	65808,3
		Итого по статье $C_{зп}$	84436,0

4.5.4.5 Расчёт отчислений во внебюджетные фонды

В данной статье расходов отражаются обязательные отчисления по установленным законодательством Российской Федерации нормам органам государственного социального страхования (ФСС), пенсионного фонда (ПФ) и медицинского страхования (ФФОМС) от затрат на оплату труда работников.

На 2018 г. в соответствии со ст. 425, 426 НК РФ действуют следующие тарифы страховых взносов: ПФР — 0.22 (22%), ФСС РФ — 0.029 (2,9%), ФФОМС — 0,051 (5,1%). Рассчитанные отчисления представлены в таблице 16.

Таблица 16. Отчисления во внебюджетные фонды

Исполнитель	**Зарплата, руб.**	**Отчисления, руб.**			**Сумма отчислений, руб.**
		ПФР (22%)	ФСС (2,9%)	ФФОМС (5,1%)	
Научный руководитель	4002,5	880,6	116,1	204,1	1200,8
Руководитель от предприятия	14625,2	3217,5	424,1	745,9	4387,5
Студент	65808,3	14477,8	1908,4	3356,2	19742,4
				Итого по статье $C_{внеб}$	25330,7

4.5.4.6 Расчёт накладных расходов

Накладные расходы учитывают все затраты, не вошедшие в предыдущие статьи расходов: печать и ксерокопирование, оплата электроэнергии, оплата пользования услугами интернета.

Перечисленные расходы требуют низких затрат денежных средств относительно заработной платы исполнителей, поэтому величина коэффициента накладных расходов $k_{накл}$ была принята в размере 5%.

Расчёт накладных расходов ведётся по формуле: $C_{накл} = k_{накл} * C_{зп}$ (5). Таким образом, $C_{накл} = 0,05 * 84436,0 = 4221,8$.

4.5.4.7 Формирование бюджета проекта

Согласно произведённым расчётам, сумма затрат по всем статьям расходов была рассчитана и представлена в качестве общего бюджета проекта в таблице 17.

Таблица 17. Бюджет проекта

Статья затрат	**Сумма, руб.**
Материальные затраты	25000
Заработная плата исполнителей	84436,0
Отчисления во внебюджетные фонды	25330,7
Накладные расходы	4221,8
Итого	138988,5

Рассчитанный бюджет не превышает бюджета в 14000 рублей, указанного в ограничениях проекта (табл. 8).

4.6 Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования

Определение эффективности производится путем определения интегрального показателя эффективности научного исследования через нахождение величин финансовой и ресурсной эффективности.

Интегральный финансовый показатель определяется по следующе формуле:

$$I_{\text{финр}}^{исп.i} = \frac{\Phi_{\text{p}i}}{\Phi_{\max}} \quad (1)$$

где $I_{\text{финр}}^{\text{исп.i}}$ — интегральный финансовый показатель разработки; $\Phi_{\text{p}i}$ — стоимость *i*-го варианта исполнения; $\Phi_{\max}$ — максимальная стоимость исполнения научно-исследовательского проекта.

Таким образом, интегральный финансовый показатель данной разработки, выполненной по варианту исполнения 1:

$I_{\text{финр}} = 138988,5 / 140000 = 0,993.$

Интегральный показатель ресурсоэффективности определяется по формуле:

$$\text{I}_{\text{pi}} = \sum_{i}^{n} a_i b_i \quad (2)$$

где I_{pi} — интегральный показатель ресурсоэффективности для i-го варианта исполнения разработки; a_i — весовой коэффициент i-го варианта исполнения разработки; b_i^p — бальная оценка i-го варианта исполнения разработки, которая устанавливается экспертным путём по выбранной шкале оценивания; n — число параметров сравнения.

Бальная оценка каждого из перечисленных ранее вариантов исполнения по техническим критериям представлена в таблице 18.

Таблица 18. Сравнительная оценка характеристик вариантов исполнения проекта

Критерии	**Весовой коэффициент**	**Баллы**			**Конкурентоспособность**		
		$И_ф$	**$И_2$**	**$И_3$**	**$И_ф$**	**$И_2$**	**$И_3$**
1. Повышение производительности труда пользователя	0,13	5	5	5	0,65	0,65	0,65
2. Удобство в эксплуатации (соответствует требованиям потребителей)	0,14	5	4	3	0,7	0,56	0,42
3. Надежность	0,15	5	4	4	0,75	0,6	0,6
4. Безопасность	0,16	5	5	5	0,8	0,8	0,8
5. Потребность в ресурсах памяти	0,07	5	4	4	0,35	0,28	0,28
6. Функциональная мощность (предоставляемые возможности)	0,12	5	5	5	0,6	0,6	0,6
7. Простота эксплуатации	0,11	5	4	4	0,55	0,44	0,44
8. Качество интеллектуального интерфейса	0,12	4	4	4	0,48	0,48	0,48
Ресурсоэффективность	**1**				**4,88**	**4,41**	**4,27**

Согласно расчётам, приведённым в таблице 18, можно сделать вывод, что самым эффективным исполнением с точки зрения ресурсоэффективности является первое, фактическое, исполнение $И_ф$ с показателем 4,88, по которому и был реализован проект в рамках магистерской диссертации.

5 СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ

5.1 Введение

Производственная и экологическая безопасность является обязательным критерием при реализации любых проектов. Раздел «Социальная безопасность» посвящён подробному разбору и анализу вопросов производственной и экологической безопасности на рабочем месте.

В данном разделе указаны такие вредные факторы, оказывающие негативное влияние на организм человека, как электромагнитное излучение, неоптимальный микроклимат помещения, высокий уровень шума и электрический ток. Так же указан характер вредного воздействия данных факторов на организм и последствия их длительного или чрезмерного воздействия. Кроме того, были указаны ЧС, которые могли произойти на рабочем месте и действия, которые необходимо выполнить в случае их возникновения.

Выполнение работы в рамках выполнения данной магистерской диссертации заключается в разработке и настройке программного обеспечения для реализации автоматизированного тестирования и мониторинга удалённых серверов VPN-сервиса. Таким образом, работу можно классифицировать как работу разработчика программного обеспечения (специалиста по автоматизированному тестированию).

Деятельность осуществлялась на территории работодателя, в офисе формата Open Space, за настольным персональным компьютером. Работодателем является компания ООО «Сибэдж», которая специализируется на заказной разработке программного обеспечения.

5.2 Производственная безопасность

Для рассмотрения производственной безопасности проекта необходимо выявить вредные и опасные факторы, которые могут возникнуть на рабочем месте, и описать мероприятия по защите исследователя и

пользователей конечных продуктов от действия этих факторов. Факторы, влияющие на виды работ с компьютером, приведены в таблице 19.

Таблица 19 — Факторы, влияющие на виды работ с компьютером

Наименование видов работ и параметров производственного процесса	**Факторы (ГОСТ 12.0.003-74 ССБТ)**	**Нормативные документы**
Вредные факторы		
Работа с компьютером	Повышенная или пониженная температура воздуха рабочей зоны	СанПиН 2.2.4.548-96
	Повышенная или пониженная влажность воздуха	СанПиН 2.2.4.548-96
	Повышенный уровень шума на рабочем месте	СанПиН 2.2.4/2.1.8.562-96
	Недостаточная освещенность рабочей зоны	СанПиН 2.2.1/2.1.1.1278-03
	Повышенный уровень электромагнитных излучений	СанПиН 2.2.2/2.4.1340-03
Опасные факторы		
Работа с компьютером	Опасность поражения электрическим током	ГОСТ 12.1.038-82
	Пожаровзрывоопасность	ГОСТ 12.1.041-83

5.2.1 Отклонение показателей микроклимата в помещении

Обеспечение в помещениях нормальных метеорологических условий является одним из необходимых условий труда, которые оказывают значительное влияние на тепловое самочувствие человека.

Нормативный документ, который отвечает за гигиенические требования к микроклимату производственных помещений — СанПиН 2.2.4.548-96 [35]. Данный нормативный документ нормирует показатели микроклимата на рабочих местах всех видов производственных помещений.

В таблице 20 отображены фактические, оптимальные и допустимые параметры микроклимата на рабочем месте.

Таблица 20 — Параметры микроклимата на рабочем месте

Период года	**Кат. раб.**	**Температура воздуха, ^{0}C**			**Относительная влажность воздуха, %**			**Скорость движения воздуха, м/с**		
		Факт.	Опт.	Доп.	Факт.	Опт.	Доп.	Факт.	Опт.	Доп.
Холодный	Iа	23	22-24	20-25	55	40-60	15-75	0,1	0,1	0,1
Теплый	Iа	24	23-25	21-28	55	40-60	15-75	0,1	0,1	0,1-0,2

Оптимальные микроклиматические условия являются предпочтительными на рабочих местах и создают условия для высокого уровня работоспособности человека. Оптимальные микроклиматические условия характеризуются тем, что эти условия обеспечивают полный комфорт тепловому и функциональному состоянию организма человека и не вызывают отклонений в состоянии здоровья человека.

5.2.2 Повышенный уровень шума на рабочем месте

Повышенный уровень шума является наиболее распространенным вредным фактором на рабочем месте, который воздействует не только на органы слуха, но и на весь организм человека через центральную нервную систему. Под действием шума ухудшается речевая коммуникация человека, снижается его реакция, а также проявляется усталость.

Источниками шума на рабочем месте оператора являются принтеры, вентиляторы систем охлаждения, множительная техника, осветительные приборы дневного света, а также шумы, проникающие извне.

Уровень шума на рабочих местах не должен превышать значений, установленных СанПиН 2.2.4/2.1.8.562-96 [36] и составлять не более 50 дБА.

Снижение уровня шума обеспечивается путём использования малошумного оборудования, звукопоглощающих материалов (плиты, панели), подвесных акустических потолков.

5.2.3 Недостаточная освещенность рабочей зоны

Недостаточная освещённость рабочей зоны оказывает негативное влияние на зрительную систему человека. Другими словами, вызывает усталость центральной нервной системы, снижает концентрацию внимания, что ведет к снижению производительности труда.

Уровень освещения на поверхности рабочего стола в зоне размещения документа, согласно СанПиН 2.2.2/2.4.1340-03 [33], должен быть в диапазоне от 300 до 500 лк. Уровень освещенности экрана не должен превышать 300 лк. Яркость осветительных приборов, находящихся в поле зрения не должна превышать 200 кд/м^2.

5.2.4 Повышенный уровень электромагнитных полей

Компьютерная техника, как любой электрический прибор, производит электромагнитное излучение.

В таблице 21 представлены временные допустимые уровни электромагнитных полей, создаваемых компьютерами на рабочих местах согласно СанПиНу 2.2.2/2.4.1340-03 [33].

Таблица 21 — Временные допустимые уровни электромагнитных полей

Наименование параметров		**Временные допустимые уровни электромагнитных полей**
Напряженность электрического поля	в диапазоне частот 5 Гц-2 кГц	25 В/м
	в диапазоне частот 2 кГц-400 кГц	2,5 В/м
Плотность магнитного потока	в диапазоне частот 5 Гц-2 кГц	250 нТл
	в диапазоне частот 5 кГц-400 кГц	25 нТл
Поверхностный видеомонитора	электростатический потенциал экрана	500В

5.2.5 Электрический ток

Поражение электрическим током является одним из опасных факторов на рабочем месте. Опасность поражения электрическим током определяется величиной тока проходящего через тело человека I или напряжением прикосновения U. Напряжение считается безопасным при напряжении прикосновения U < 42 В.

Результатом воздействия на организм человека электрического тока могут быть электротравмы, электрические удары и даже смерть (согласно ГОСТ 12.1.009-2009 [39]).

Для того, чтобы защититься от поражения электрическим током, необходимо:

- обеспечить недоступность токоведущих частей от случайных прикосновений;
- электрическое разделение цепи;
- устранить опасности поражения при проявлении напряжения на разных частях.

Таблица 22 отображает предельно допустимые значения напряжения прикосновения и тока на рабочем месте (согласно ГОСТ 12.1.038-82 [40]).

Таблица 22 — Допустимые значения напряжения прикосновения и тока

Род тока	**Напряжения прикосновения, В**	**Ток, мА**
	Не более	
Переменный, 50 Гц	2,0	0,3
Постоянный	8,0	1,0

По электробезопасности рабочее место относится к помещениям без повышенной опасности поражения людей электрическим током. Данный фактор характеризуется отсутствием условий, создающих повышенную или особую опасность.

5.3 Экологическая безопасность

Экологическая безопасность и охрана окружающей среды являются одними из важнейших факторов при выполнении работ любого характера. При работе в офисном помещении за персональным компьютером отсутствуют выбросы в окружающую среду и нет влияния на жилищную зону.

Поскольку при разработке данной магистерской диссертации использовался компьютер, необходимо помнить о правильной утилизации компьютерного лома после выхода из строя данного ПК. В соответствии с постановлением правительства №340 [38] юридическим лицам запрещено самостоятельно утилизировать компьютерную технику. Необходимо найти организацию, которая занимается утилизацией в частном порядке.

В нормативном документе СанПиН 2.2.2/2.4.1340-03 [33], даются следующие общие рекомендации по снижению опасности для окружающей среды, исходящей от компьютерной техники:

- применять оборудование, соответствующее санитарным нормам и стандартам экологической безопасности;
- применять расходные материалы с высоким коэффициентом использования и возможностью их полной или частичной регенерации;
- отходы в виде компьютерного лома утилизировать;
- использовать экономные режимы работы оборудования.

5.4 Безопасность в чрезвычайных ситуациях, общие правила поведения в чрезвычайных ситуациях

Наиболее вероятная чрезвычайная ситуация при работе в офисе — это пожар. Рабочее место относится к категории «В» (пожароопасные), потому что в данном помещении присутствует пыль, вещества и материалы, способные при взаимодействии с воздухом только гореть.

Возникновение пожара может быть обусловлено следующими факторами:

- возникновением короткого замыкания в электропроводке вследствие неисправности самой проводки или электросоединений и электрораспределительных щитов;
- возгоранием устройств вычислительной аппаратуры вследствие нарушения изоляции или неисправности самой аппаратуры;
- возгоранием мебели или пола по причине нарушения правил пожарной безопасности, а также неправильного использования дополнительных бытовых электроприборов и электроустановок;
- возгоранием устройств искусственного освещения.

Для устранения возможных причин возникновения пожаров необходимо проводить следующие мероприятия:

1. Организация мероприятия:

- противопожарный инструктаж обслуживающего персонала;
- обучение персонала техники безопасности;
- разработка инструкций, планов эвакуаций и т.п.

2. Эксплуатационные мероприятия:

- соблюдение эксплуатационных норм оборудования;
- выбор и использование современных автоматических средств пожаротушения.

3. Технические мероприятия:

- профилактический осмотр и ремонт оборудования;
- соблюдение противопожарных мероприятий при устройстве электропроводок, оборудования, систем отопления и т.п.

5.5 Правовые и организационные вопросы обеспечения безопасности

5.5.1 Правовые нормы трудового законодательства

Продолжительность рабочего дня не должна превышать 40 часов в неделю. Вид трудовой деятельности за компьютерным устройством (компьютер, мобильное устройство), в рамках выполнения выпускной квалификационной работы, соответствует группе «В» — творческая работа в режиме диалога с компьютерным устройством. Категория данной трудовой деятельности соответствует III (до 6 часов непосредственной работы за компьютером).

Продолжительность непрерывной работ за компьютерным устройством, без регламентированного перерыва, не должна превышать 2 часа. Длительность регламентированных перерывов составляет 20 минут (после 1,5-2,0 часа от начала рабочей смены и обеденного перерыва). Также, необходимо уделять время нерегламентированным перерывам (микропаузы), длительность которых составляет 1-3 минуты.

5.5.2 Требования к организации и оборудованию рабочих мест

Рабочее место — это часть рабочей зоны. Оно представляет собой место постоянного или временного пребывания работника в процессе трудовой деятельности.

В соответствии с ГОСТ 12.2.032-78 ССБТ «Рабочее место при выполнении работ сидя. Общие эргономические требования» [37] к рабочему месту предъявляются следующие основные требования:

- Конструкцией рабочего места должно быть обеспечено выполнение трудовых операций в пределах зоны досягаемости моторного поля.
- При организации рабочего места следует учитывать антропометрические показатели женщин (если работают только женщины) и

мужчин (если работают только мужчины); если работают и женщины, и мужчины — общие средние показатели женщин и мужчин.

- Конструкцией рабочего места должно быть обеспечено оптимальное положение работающего, которое достигается регулированием высоты рабочей поверхности, сиденья и пространства для ног.

Заключение

В процессе исследования был произведён анализ предметной области, приведено обоснование необходимости реализации автоматизированного тестирования, выполнен анализ подходов и средств реализации автоматизированного тестирования.

В результате исследования был выбран подходящий способ реализации решений для автоматизированного тестирования и мониторинга удалённых серверов VPN-сервиса и для автоматизированного тестирования пользовательского интерфейса SDK-приложения.

Для тестирования удалённых серверов VPN-сервиса был разработан фреймворк на языке программирования Python, объединяющий в себе параметризацию запуска тестов, реализацию тестовых сценариев, обработку и экспорт результатов тестирования в систему сбора данных Prometheus. Для визуализации и мониторинга результатов тестирования по одному из тестовых сценариев была разработана статистическая страница в системе Grafana. Для сборки и запуска тестов по расписанию была выполнена интеграция с системой непрерывной интеграции CircleCI.

Для автоматизации регрессионного тестирования SDK-приложения были реализованы автоматизированные тесты пользовательского интерфейса на языке программирования Java. При реализации были использованы такие инструменты, как Espresso, UI Automator, Barista, JUnit, паттерн проектирования Testing Robots и сервис Firebase Test Lab.

Реализованные тестовые решения внедрены и используются в процессах разработки и поддержки VPN-сервиса и сопутствующего программного обеспечения в компании ООО «Сибэдж».

Автоматизация позволила снизить ресурсозатраты, повысить эффективность и скорость выполнения процесса тестирования.

Список используемых источников

1. Новичков А., Панкратов В. Автоматизированное тестирование: оценка возврата инвестиций и сопутствующие риски. — КомпьютерПресс, 2005. — №11. — С. 47-49.

2. Росляков А. Виртуальные частные сети VPN. Модели и методы анализа. — LAP Lambert Academic Publishing, 2011. — С. 328.

3. ISTQB Exam Certification [Электронный ресурс]: «ISTQB Exam Study Material». — Режим доступа: http://istqbexamcertification.com/ (дата обращения: 01.05.2018).

4. International Organization for Standardization [Электронный ресурс]: «ISO 9000:2015. Quality management systems — Fundamentals and vocabulary». — Режим доступа: https://www.iso.org/standard/45481.html (дата обращения: 01.05.2018).

5. International Organization for Standardization [Электронный ресурс]: «ISO/IEC 25000:2014. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE». — Режим доступа: https://www.iso.org/standard/64764.html (дата обращения: 02.05.2018).

6. Электронный фонд правовой и нормативно-технической документации [Электронный ресурс]: «ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств». — Режим доступа: http://docs.cntd.ru/document/gost-r-iso-mek-12207-2010 (дата обращения: 03.05.2018).

7. International Organization for Standardization [Электронный ресурс]: «ISO/IEC TR 19759:2015. Software Engineering — Guide to the software engineering body of knowledge (SWEBOK)». — Режим доступа: https://www.iso.org/standard/64764.html (дата обращения: 03.05.2018).

8. Савин Р. тестирование dot com или Пособие по жестокому обращению с багами в интернет-стартапах. — М.: Издательство «Дело», 2007. — 312 с.

9. Лиза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. — М.: «Вильямс», 2010. — 464 с.

10. Про Тестинг [Электронный ресурс]: «Автоматизированное тестирование программного обеспечения — основные понятия». — Режим доступа: http://www.protesting.ru/automation/ (дата обращения: 04.05.2018).

11. Куликов С. Тестирование программного обеспечения. Базовый курс. — Минск: «Четыре четверти», 2015. — 293 с.

12. Майк Кон. Scrum. Гибкая разработка ПО. — М.: «Вильямс», 2011. — 576 с.

13. SibEDGE. Качественное программное обеспечение [Электронный ресурс]. — Режим доступа: http://sibedge.com/ru (дата обращения: 20.03.2018).

14. Хабрахабр. Блог компании Badoo [Электронный ресурс]: «Путеводитель по инструментам автотестирования мобильных приложений». — Режим доступа: https://habr.com/company/badoo/blog/347986/#selendroid_robotium (дата обращения: 10.03.2018).

15. Appium. Automation for Apps [Электронный ресурс]. — Режим доступа: http://appium.io/ (дата обращения: 10.03.2018).

16. Calabash. Automated acceptance testing for mobile apps [Электронный ресурс]. — Режим доступа: https://calaba.sh/ (дата обращения: 10.03.2018).

17. Android Developers. Test apps on Android [Электронный ресурс]. — Режим доступа: https://developer.android.com/training/testing/ (дата обращения: 10.03.2018).

18. Анмеш Гандеча. Selenium Testing Tools Cookbook. — Packt Publishing, 2012. — 326 с.

19. Марк Саммерфилд. Python на практике. — М.: ДМК Пресс, 2014. — 338 с.

20. CircleCI: Continuous Integration and Delivery [Электронный ресурс]. — Режим доступа: https://circleci.com/docs/2.0/ (дата обращения: 01.02.2018).

21. Python Documentation [Электронный ресурс]: «argparse — Parser for command-line options, arguments and sub-commands». — Режим доступа: https://docs.python.org/3/library/argparse.html (дата обращения: 01.02.2018).

22. GitHub [Электронный ресурс]: «ParallelSSH». — Режим доступа: https://github.com/ParallelSSH/parallel-ssh (дата обращения: 01.02.2018).

23. Prometheus: Monitoring system & time series database [Электронный ресурс]. — Режим доступа: https://prometheus.io/docs/introduction/overview/ (дата обращения: 01.02.2018).

24. GitHub [Электронный ресурс]: «client_python. Prometheus instrumentation library for Python applications». — Режим доступа: https://github.com/prometheus/client_python (дата обращения: 01.02.2018). [Электронный ресурс]. — Режим доступа: https://circleci.com/ (дата обращения: 01.02.2018).

25. Grafana: The open platform for analytics and monitoring [Электронный ресурс]. — Режим доступа: http://docs.grafana.org/ (дата обращения: 01.02.2018).

26. Jake Wharton [Электронный ресурс]: «Testing Robots». — Режим доступа: https://jakewharton.com/testing-robots/ (дата обращения: 05.02.2018).

27. GitHub [Электронный ресурс]: «Barista». — Режим доступа: https://github.com/SchibstedSpain/Barista (дата обращения: 07.02.2018).

28. JUnit [Электронный ресурс]. — Режим доступа: https://junit.org/junit4/ (дата обращения: 01.02.2018).

29. Frebase Test Lab [Электронный ресурс]. — Режим доступа: https://firebase.google.com/docs/test-lab/ (дата обращения: 01.02.2018).

30. Издательство «Открытые системы» [Электронный ресурс]: «Средства автоматизированного тестирования». Режим доступа: https://www.osp.ru/os/2009/03/8161608/ (дата обращения: 03.09.2017).

31. IBM [Электронный ресурс]: «Rational Functional Tester». Режим доступа: https://www.ibm.com/us-en/marketplace/rational-functional-tester (дата обращения: 03.09.2017).

32. AutoIt [Электронный ресурс]: «AutoIt Scripting Language». Режим доступа: https://www.autoitscript.com/site/autoit/ (дата обращения: 03.09.2017).

33. СанПиН 2.2.2/2.4.1340-03. Санитарно-эпидемиологические правила и нормативы. Гигиенические требования к персональным электронно-вычислительным машинам и организации работы.

34. СанПиН 2.2.4.548-96. Гигиенические требования к микроклимату производственных помещений.

35. СанПиН 2.2.4.1191-03. Электромагнитные поля в производственных условиях.

36. СанПиН 2.2.4/2.1.8.562-96. Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории застройки.

37. ГОСТ 12.2.032-78. Рабочее место при выполнении работ сидя. Общие эргономические требования.

38. Постановление правительства РФ №340. О порядке установления требований к программам в области энергосбережения и повышения энергетической эффективности организаций, осуществляющих регулируемые виды деятельности».

39. ГОСТ Р 12.1.009-2009 Система стандартов безопасности труда (ССБТ). Электробезопасность. Термины и определения.

40. ГОСТ 12.1.038-82 Система стандартов безопасности труда (ССБТ). Электробезопасность. Предельно допустимые значения напряжений прикосновения и токов.

Приложение А

(справочное)

Automation testing and monitoring of VPN-service remote servers

Студент:

Группа	ФИО	Подпись	Дата
8ИМ6А	Назимок Павел Павлович		

Руководитель ВКР:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Шестаков Н.А.	к.т.н.		

Консультант отделения информационных технологий ИШИТР:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Мирошниченко Е.А.	к.т.н.		

Консультант-лингвист отделения иностранных языков ШБИП:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИЯ	Комиссарова О.В.	к.ф.н.		

A1 Test automation

In recent years, the view of software testing has matured into a constructive one. Testing is no longer seen as an activity that starts only after the coding phase is complete with the limited purpose of detecting failures. Software testing is or should be pervasive throughout the entire development and maintenance life cycle. Indeed, planning for software testing should start with the early stages of the software requirements process, and test plans and procedures should be systematically and continuously developed as software development proceeds [1]. As a result, the probability of identifying potential and actual errors at an early stage is significantly increased, which positively affects the quality of the entire product, since the error was identified by the developer, and not by the customer.

In order to reduce the expenses and increase the efficiency of the testing process various specialized testing methods are in use, the testers cooperate together with developers and there is a constant communication with the customer. Another way is to replace manual testing with an automated one.

A1.1 The meaning of test automation

In reality, testing is done by performing a certain set of actions on the application under test, obtaining the results of these actions and their further analysis. This process can be performed by QA engineers either manually or automatically using various software tools. The process of software verification, during which the basic functions and test steps are performed automatically using tools for automated testing, is called software test automation [2].

Test automation has its own advantages and disadvantages. The pros of test automation are:

- Fast execution speed, far superior to human capabilities.
- Lack of the «human factor» (inattention, fatigue).
- Possibility of repeated test execution, which allows reducing testing process costs.

- Execution of complex test cases, which cannot be done manually.
- The ability to store and analyze large amounts of test data.

The cons of test automation are:

- The need of a highly qualified QA automation engineer;
- Additional expenses for automation tools, test development and test maintenance.
- Certain tests should be rewritten in case of UI or requirements changes.

There are certain types of testing, which makes the use of test automation particularly beneficial.

A2.1 The use of test automation

Test automation does not completely exclude manual testing from the development process, but significantly reduces its percentage and costs by eliminating routine operations from the testing process.

There are two common routine testing processes, which are most often automated by QA engineers [3].

1. **Smoke Testing** is a type of software testing that includes a non-exhaustive set of tests designed to test the most important functions. The results of this testing are used to decide whether the build is stable enough to continue testing, or to verify the correct functioning of the system [3].

2. **Regression Testing** is a collective name for all types of software testing aimed at detecting errors in already tested areas of the source code. Regression testing checks the product for errors that could appear as a result of adding a new functionality to the program or fixing tracked errors. The purpose of this type of testing is to make sure that software updates or bug fixing have not caused brand new bugs [3].

Today it is difficult to imagine the development of a large software project without using automated testing in the development process. The current state of

testing is characterized by deep integration with the development process as a whole, as well as the widest use of automation along with variety automation tools [4].

This master's thesis is devoted to automation of smoke and regression functional testing of different parts of one system at the integration and system levels, respectively.

A3.1 Automation testing pyramid

There are three types of automated tests that are commonly used in development [5].

1. **Unit Tests**. Implemented by developers to test individual parts of the program code.
2. **Integration, functional tests (Service Tests)**. Testing with the access to the functional layer, bypassing the user interface. A quick way to test system components. These tests can be run without user efforts and check the system functionality at a high speed.
3. **UI-tests**. Implement end-to-end coverage and ensure stable state of the system and critical business requirements through interaction with the user interface.

Mike Cohn organized these groups of tests into a hierarchy, known as the «Testing pyramid» (figure 18), which illustrates the correlation of the tests needed to effectively test the system. To date, the concept of the classical pyramid is the standard in the QA industry [5].

There is the level of manual testing at the top of the pyramid, as it is somehow always present and executed, at least for acquaintance with the system and its updated functionality.

It is important to note the similarity of the hierarchy of automated testing with the hierarchy of all testing in general, which was mentioned earlier in section 1.3.1. The difference is that during automation, System testing is done by interaction with the user interface and comparing the software behavior with the expected one.

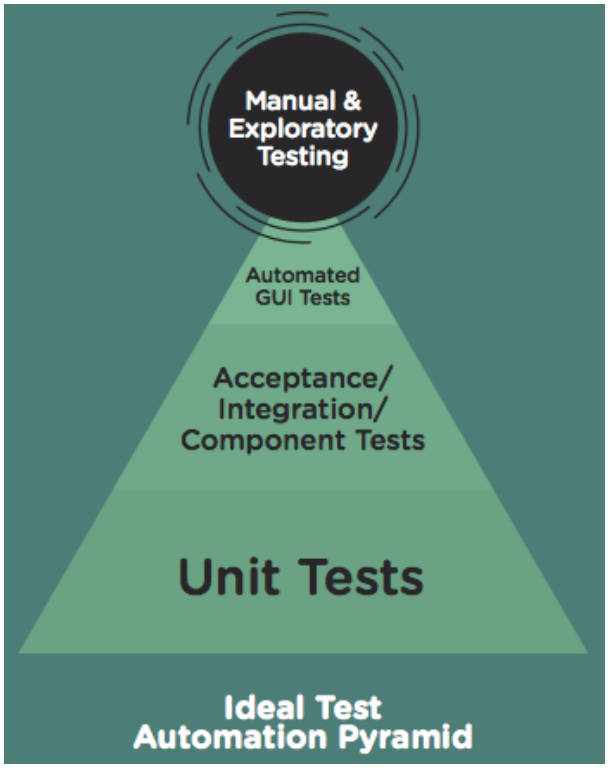

Figure 18. Testing automation pyramid

A2 Test automation tools

This master's thesis is devoted to the implementation of automated UI tests (the upper level of the pyramid) of a mobile Android application, as well as solution for performing automated functional tests bypassing the user interface (the middle level of the pyramid) and collecting their results. These types of automation are implemented using specialized tools for test automation that provide the ability to interact with the interface of the software under test.

There are many tools for implementing automated tests. Each of them is aimed to solve different tasks in terms of complexity, size, execution environments and many other characteristics. The following describes the tools that are suitable for solving the current thesis' problems.

A1.2 Testing and monitoring of remote servers

The development of a solution for automated testing and monitoring of remote servers includes such important details as integration with the Prometheus data collection and notification system and the creation of an SSH connection

followed by the execution of certain commands on the remote server. The implementation of these capabilities, given that the SSH connection must be created with more than 1000 servers, is feasible only at the software level. Taking into account the requirements to the software solution, no ready-made tools for completing this task were found. In this situation, the right thing to do is to develop your own solution for testing.

Such solutions are usually called frameworks. There are many different definitions of the framework. The most suitable and widely used in the QA industry: «The framework for test automation is a set of interacting components that facilitate the creation and execution of automated tests and the presentation of their results» [6]. In other words, the framework for test automation covers everything a QA specialist needs to write, run and analyze automated tests. It can also include the implementation of the tests themselves.

A2.2 Testing of SDK application

During the test process, various automation tools manipulate the application. The application itself interacts with the user, the system and other applications through such interfaces as:

- API (Application Programming Interface) — the main interface for interaction with other programs.
- GUI (Graphic User Interface) — a graphical user interface to interact with the user.
- Net (Networking Interface) — network interface to interact with the Internet.

Automated tests can use all these interfaces to interact with the application. For manual testing, the agent between the tests and the application is the manual test engineer: he converts the text of the test cases into actions and applies it to application and its interfaces (figure 19) [7].

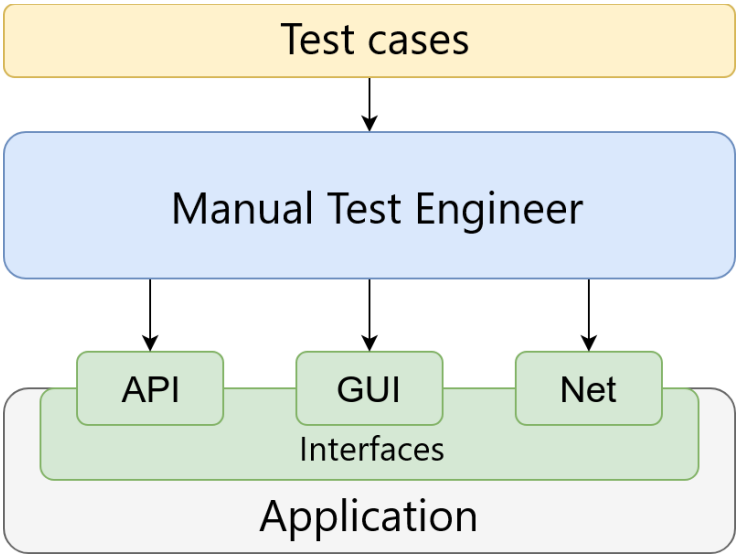

Figure 19. Mobile application manual testing scheme

For automation, you need to replace the manual tester with tools that can interact with one or more application interfaces. In addition, you need utilities to execute a set of test cases (figure 20) [7].

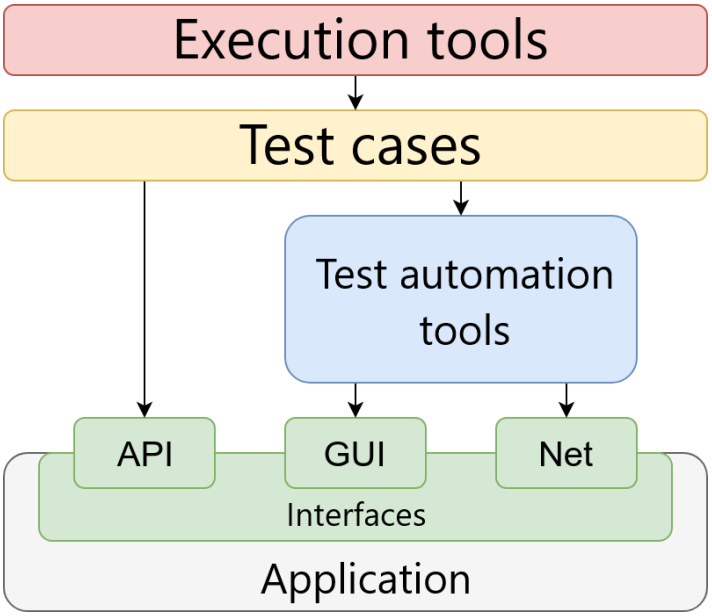

Figure 20. Mobile application automated testing scheme

The SDK application under test is developed for the Android platform, so here is an overview of the tools that are suitable for GUI test automation of Android applications.

A2.2.1 Appium

Appium is an open source free mobile automation tool offering automation of native and hybrid mobile apps [7].

Some advantages of using the Appium testing framework:

- It is a free to use open source tool used for automating both Android and iOS platform apps.
- It supports a number of programming languages such as Java, C#, and Ruby without the need to modify apps for automation purposes.
- Appium is a «cross-platform» tool enabling substantial code reuse between iOS and Android test suites.
- It runs on multiple devices and emulators offering scalability.

The downside is that it offers no support for toast messages and has slow scripting. Additionally, some testing professionals feel that the test reports generated are poor and inadequate.

A2.2.2 Calabash

Calabash is yet another free open source mobile automation tool for Android and iOS mobile apps. Calabash testing framework allows you to write and execute tests that validate the functionality of iOS and Android apps. It is written in Ruby [7].

Some advantages of using Calabash testing framework:

- It is a free open source tool used for automating and testing any Android and iOS platform apps, native or hybrid, from the UI level down.
- It is a cross platform tool.
- Calabash supports testing on real devices as well as on simulators.
- Calabash allows support for complex features as gestures — like tap, swipe and rotate, and generate better reports than Appium.

Some of the cons include lack of adequate online support, non-availability of an IDE or an Editor and unfriendliness to languages other than Ruby.

A2.2.3 Espresso

It is Google's open source test automation framework for native Android apps. Espresso is inside the application and has been written by Google primarily for functional testing of an Android UI. It is mainly targeted for developers or QA automation engineers [7].

Some advantages of using Espresso testing framework:

- The tool is simple to setup and supports all Android versions.
- Its library is supported and maintained by Google community.
- Espresso does not need a server to communicate with, instead, it runs simultaneously with the app and delivers quick test results to the developer.
- Espresso allows for less testing flakiness and better test cycle stability with its unique synchronization method.
- With Espresso, extensive testing is possible as it covers a number of UI actions and gestures.

Some of the cons include lack of cross platform support, and besides, it is not a dev-language free framework like Appium and is an app-context only tool.

A2.2.4 UI Automator

It is an open source testing framework by Google that offers advanced UI testing of native Android apps and games. It uses Java as the programming language and is considered as one of the best tools available for Android apps [7].

Some advantages of using UI Automator framework:

- Scripts created can be executed on many different Android platforms.
- Its library is supported and maintained by Google community.
- UI Automator allows test cases to run on real Android devices, and its light-weight API is easy to adopt and use for the native apps.

Some of the cons include restricted language support with test scripting only in Java and lack of internal view access.

A2.2.5 Tools overview conclusions

The most widely used tool for implementing automated testing from both the reviewed and all existing ones at the moment is Appium [7]. This is due to the fact that it is cross-platform, which means that you can reuse the same automated tests in both iOS and Android applications. In addition, it supports a large number of programming languages in comparison with other tools. Two of these features make it the most popular tool for test automation.

In addition, considering that the SDK application under test runs on the Android platform, the tools that are aimed only on Android testing were also reviewed. So, the Espresso framework, takes into account the various features of Android, which makes it much easier to write automated tests. However, when using it or similar tools, the test base for iOS and Android applications will be different, so that you have to perform double amount of work both in the part of implementation and maintenance of test automation.

The choice of tool to implement with is described in Section 3.1.2.

References

1. International Organization for Standardization [Electronic source]: «ISO/IEC TR 19759:2015. Software Engineering — Guide to the software engineering body of knowledge (SWEBOK)». — URL: https://www.iso.org/standard/64764.html (access date: 03.05.2018).

2. Pro Testing [Electronic Source]: «Software test automation — the basic concepts». — URL: http://www.protesting.ru/automation/ (access date: 04.05.2018).

3. R. Savin. Testing dot com or Guide to bully bugs in Internet Startups. — M.: Publishing house «Delo», 2007. — 312 p.

4. S. Kulikov. Software testing. Basic course. — Minsk: «Chetire chetverti», 2015. — 293 p.

5. Mike Cohn. Succeeding with Agile: Software Development Using Scrum. — M.: «Williams», 2011. — 576 p.

6. Lisa Crispin, Janet Gregory. «Agile Testing: A Practical Guide for Testers and Agile Teams». — M.: «Williams», 2010. — 464 p.

7. Habr. Badoo blog [Electronic source]: «Test automation tools for mobile applications guide». — URL: https://habr.com/company/badoo/blog/347986/#selendroid_robotium (access date: 10.03.2018).