федеральное государственное автономное образовательное учреждение
ВЫСШЕГО ОБРАЗОВАНИЯ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет *Информационных Технологий и Программирования*
Направление (специальность) *Прикладная математика и информатика*
Квалификация (степень) *Бакалавр прикладной математики и информатики*
Кафедра *Компьютерных технологий* Группа *4539*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Разработка методики автоматического тестирования взаимодействий мобильных приложений с системой Android

Автор квалификационной работы *Баглай Б.* (подпись)

Руководитель *Макаров Н. С.* (подпись)

Консультанты:

а) По экономике и организации производства _____ (подпись)

б) По безопасности жизнедеятельности и экологии _____ (подпись)

в) _____ (подпись)

К защите допустить

Зав. кафедрой *Васильев В.Н.* (подпись)

«___» _____ 2015 г.

Санкт-Петербург, 2015 г.

Дата защиты «____» _____ 2015 г.

Секретарь ГАК _____

Листов хранения _____

Чертежей хранения _____

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка метода автоматического тестирования взаимодействий мобильных приложений с операционной системой Android. Под взаимодействием приложения с системой, имеются в виду любые действия пользователя в операционной системе, имеющие какое-либо влияние на работу приложения и наоборот. Любые действия пользователя в приложении, влияющие на работу операционной системы.

В настоящее время широкое распространение получили приложения, взаимодействующие с системой. Так, например, системные уведомления, без которых уже трудно представить современное мобильное приложение. «Список дел» напоминает нам выполнить запланированное важное дело, а приложение «Здоровье» рекомендует сделать зарядку. Кликнув по одному из таких уведомлений, мы перейдем в приложение уведомившее нас. В какую часть приложения мы попадем, зависит от самого уведомления. Уведомления могут вести в разные части приложения, которых может быть не мало. А сами уведомления, создаются при определенных условиях: вышло обновление, вам написали сообщение, вам отправили файл. Логику работы таких уведомлений хочется полноценно тестировать перед релизом приложения. Но большое число всевозможных комбинаций уведомлений, и не малое время создания необходимых условий, очень сильно замедляет ручное тестирование данной функциональности приложения. На решение данной проблемы и нацелена эта работа.

Для достижения цели ставятся следующая задача: разработать методику автоматизированного тестирования приложений на ОС Android, объединяющую существующие подходы с целью гибкого использования преимуществ каждого из них.

Написанных на момент реализации, поставленной в данной работе, цели, в качестве традиционного автоматического тестирования мобильных приложений на Android использовалось два подхода. Эти подходы отличны друг от друга и не взаимозаменяемы по функциональности. Существующие инструменты автоматического тестирования базируются на использовании одного из этих подходов. Но, во многих случаях, в мобильном автоматическом тестировании есть необходимость в применении обоих способов. В виду такой необходимости существует инструмент — «обертка» над проектами обоих подходов. По своей сути, это два независимых инструмента. А при запуске автоматического теста, в зависимости от необходимой функциональности, запускается один из инструментов. Большой минус такой «обертки» — это отсутствие связи двух подходов на программном уровне. Кроме того, проекты, входящие в состав данного решения, разрабатываются разными командами. Поддержание работоспособности автоматизированных тестов для двух инструментов несет дополнительные временные расходы и ведет к понижению общей стабильности, и надежности системы автотестирования.

В данной работе, в рамках реализации новой методики был создан программный комплекс, объединяющий оба подхода автоматического тестирования на программном уровне, и, как следствие, решающий проблемы традиционных инструментов. Разработанный комплекс успешно прошел проверку в системе автоматизированного тестирования мобильных приложений под Android на предприятии, что подтверждено актом о внедрении. В настоящее время, данная разработка обеспечивает функционирование более тридцати автоматизированных тестов взаимодействующих с системой Android. Это существенно освободило время работников ручного тестирования для проведения более интеллектуальных задач контроля качества, что привело к повышению качества конечного продукта. Кроме того, учитывая возможности разработанного инструмента, было принято решение о покрытии автоматическими тестами около семидесяти новых тестовых сценариев, основанных на взаимодействии с системой.

Первая глава работы посвящена обзору известных систем автоматического тестирования мобильных приложений на Android. Там же обсуждаются недостатки существующих систем автоматического тестирования мобильных приложений.

Во второй главе, на основании анализа различных систем автоматического тестирования мобильных приложений, а также опыта автоматического тестирования системных взаимодействий на предприятии, сформулированы основные принципы и требования к системе. Далее приведено общее описание и модель автоматической системы тестирования системных взаимодействий.

В третьей главе приведена архитектура созданного решения. Описано взаимодействия тестовой программы с сервером тестирующего инструмента и других частей разработанной системы автоматического тестирования.

В заключении дано описание текущего состояния разработки, достигнутые результаты и перспективы развития системы.

ГЛАВА 1. Методы автоматического тестирования Android приложений

1.1. Описание предметной области

В данном разделе подробно опишем цели работы, а также введем термины, используемые в других частях представленной работы.

1.1.1. Термины и понятия

Введем термины и понятия, касающиеся данной работы. В дальнейшем приведенные термины будут использоваться в указанных значениях, если не оговорено обратное.

Системное взаимодействие — любое действие пользователя в операционной системе, имеющее какое-либо влияние на работу приложения и наоборот, любое действие пользователя в приложении, влияющие на работу операционной системы. Речь идет об обычном пользователе, который, в обоих случаях, взаимодействует с системой или приложением посредством пользовательского интерфейса.

1.1.2. Цель работы

Как уже было упомянуто в ведении, целью данной работы является разработка метода, позволяющего осуществлять автоматизированное тестирование системных взаимодействий мобильного приложения на Android. Для простоты понимания, приведем несколько примеров таких взаимодействий.

Самым простым взаимодействием пользователя с ОС, влияющим на работу приложения, может быть изменение текущего времени в настройках телефона. Приложение, состояние которого зависит от времени («Будильник», «Фазы сна» и т.д.), должно будет корректно отреагировать на данное системное взаимодействие. Подобным образом, изменения языка в системе, может повлечь соответствующие изменения в приложении.

Особый интерес представляют собой push-уведомления [6]. В Android нет никакой встроенной системы для прямого отображения пользователю push-уведомлений. Все данные передаются («пушатся») исключительно в само приложение и в произвольной форме. После получения данных, приложение может, например, выдать стандартное для Android уведомление. Такое уведомление отобразится в верхней части экрана и в “шторке”. А может появится баннер, как в iOS. Учитывая открытость и гибкость ОС Android, после получения push-уведомления, оно может выводиться, практически в любой форме [7].

В виду гибкости и экономичности к ресурсам, данная технология получила широкое распространение в современных мобильных приложениях. Функциональность приложений все чаще задействует технологию push-уведомлений. Но, тестирование функциональности, связанной с push-уведомлениями, занимает много времени работников ручного тестирования. В основном, из-за необходимости создавать определенные условия, при которых, трестируемое push-уведомление генерируется. А также из-за необходимости производить определенные манипуляции в пользовательском интерфейсе вне

трестируемого приложения, что увеличивает вероятность ошибки тестировщика. В случае ошибки все действия, проделанные для создания определенных условий, необходимо повторить заново. Учитывая, что таких уведомлений может быть десятки в одном приложении, процесс ручного тестирования данной функциональности будет очень сильно тормозить цикл разработки приложения.

Далее в работе, из системных взаимодействий, по большей части, мы будем говорить о push-уведомления, ввиду их обширного использования. Но, рассмотренные методы и разработанное решение актуальны и применимы ко всем видам системных взаимодействий.

Уточним, что наша цель — не просто иметь возможность тестировать системные взаимодействия, но делать это в условиях процесса разработки реального мобильного приложения, нацеленного на широкий круг пользователей. Это означает, что приложение должно корректно работать на довольно большом диапазоне версий Android. Что накладывает дополнительные требования к функциональности инструмента используемого в целях автоматического тестирования. Данное ограничение сильно повлияло на обзор существующих решений автоматического тестирования.

Есть еще одно немаловажное требование к системе автоматического тестирования. Система должна базироваться на использовании одного инструмента автоматического тестирования. Данное ограничение необходимо для обеспечения стабильность системы автоматического тестирования в целом. Использование нескольких инструментов влечет за собой следующие сложности:

- дополнительные трудозатраты на поддержание в работоспособном состоянии написанных автоматизированных тестов
- трудности, связанные с переходами на новые версии используемых инструментов
- может частично пропасть совместимость инструментов
- разработка разных инструментов ведется разными командами.

1.2. Анализ

В данном разделе описаны требования к инструменту автоматического тестирования приложений, написанных под операционную систему Android. Описаны существующие, на момент написания данной работы, инструменты, а так же, рассмотрены недостатки готовых решений.

1.2.1. Требования

Учитывая требования, выдвинутые в части 1.1.2., составим список из наиболее важных требований к инструменту автоматического тестирования мобильных приложений. Стоит отметить, что все рассматриваемые инструменты удовлетворяют некоторому подмножеству требований, сравнение по таким требованиям хорошо представлено в данной статье [19]. Эта статья анализирует существующие подходы к автоматизации тестирования пользовательского интерфейса в целом, структуру и основные компоненты системы Android, а также описаны основные существующие подходы автоматизированного тестирования графического пользовательского интерфейса.

Перечислим наиболее показательные требования:

- Тестовое приложение не должно изменяться.
- Поддержка старых (< 4.2) версий Android.
- Поддержка тестирования системных взаимодействий.
- Система автотестирования должна базироваться на одном инструменте.

1.2.2. Существующие решения

В виду большого количества инструментов автоматического тестирования для платформы Android, в обзор попали инструменты, максимально удовлетворяющие основным требованиям, описанным в части 1.2.1.

Все существующие решения после отсеивания по первому требованию делятся на два вида (Рисунок 1.). Первый вид, использующий подход Instrumentation [8]. Второй, использующий подход UIAutomator [9].

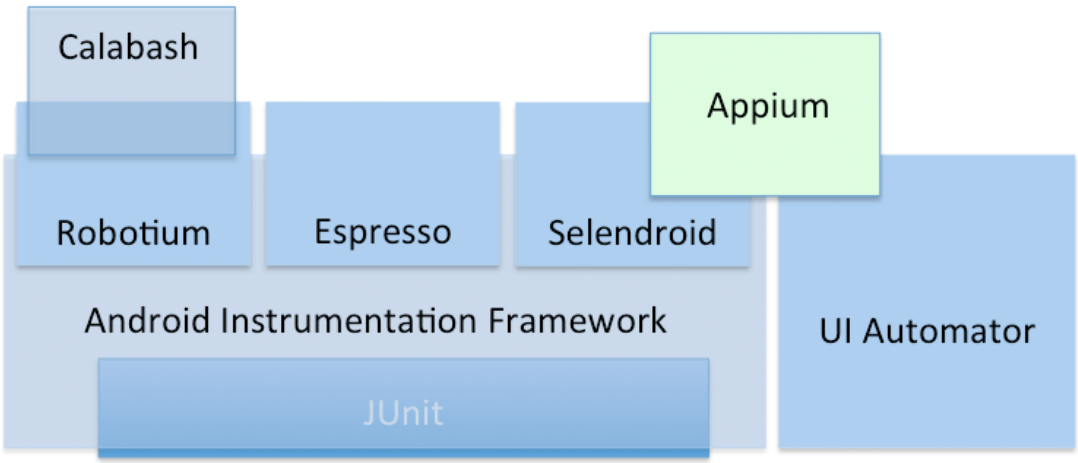


Рис. 1. Схема готовых решений и методов, которые они используют

Instrumentation — это набор методов Android системы, контролирующих загрузку приложения и Android компоненты независимо от их обычного «жизненного цикла». Поставляется вместе с Android SDK. Данный способ был изначально, API level 1 [16]. Недостатком данного подхода является отсутствие возможности тестирования объектов вне трестируемого приложения. Это означает, что с помощью инструментов, использующих данный подход, нет возможности тестировать системные взаимодействия.

UI Automator — фреймворк тестирования, предоставляющий набор API для тестирования пользовательского интерфейса трестируемого приложения и системы [17]. Поставляется вместе с Android SDK. Android SDK содержит следующие инструменты автоматического тестирования пользовательского интерфейса:

- UIAutomatorviewer — графический инструмент для поиска компонентов пользовательского интерфейса в Android приложении.
- UIAutomator — библиотеки Java API. Содержат методы для тестирования пользовательского интерфейса.

Для использования UIAutomator, необходима версия Android SDK Tools 21 или выше, API 16 или выше. Из чего следует, что UIAutomator может тестировать только приложения, версия которых выше 4.1.

1.2.2.1 Selendroid

Самым ярким представителем Instrumentation подхода является проект Selendroid [4]. Данный проект активно разрабатывается корпорацией Ebay [10]. Selendroid поддерживает возможность тестирования приложений, начиная с 10-ой версии Android API [16]. Это означает, что данный инструмент поддерживает тестирование на устройствах, с версией Android 2.3.3 «Gingerbread» и выше.

Selendroid работает с «нативными» (native) приложениями и гибридными (содержащими web объекты). Тесты пишутся с использованием клиентского API WebDriver [11]. Работает с эмуляторами и реальными устройствами, и может быть интегрирован в качестве узла в Selenium Grid [12] для масштабирования и параллельного тестирования. На схеме (Рисунок 2.) представлена архитектура проекта Selendroid.

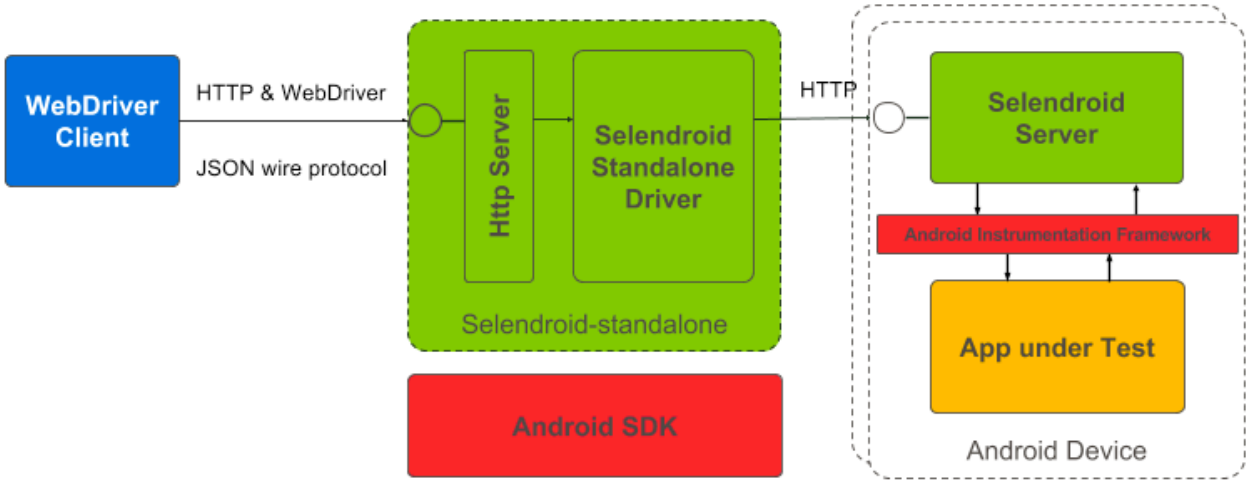


Рис. 2. Схема архитектуры Selendroid [5]

1.2.2.1 Appium

Проект Appium [13] является наиболее интересным для нас инструментом. Его направление развития подтверждает важность выдвинутых нами требований к инструменту автоматического тестирования. Данная работа [18] подробно рассматривает работу с Appium. В данной статье выявлен ряд недостатков Appium. Наиболее важный для нас недостаток — «Appium uses UIAutomator for Android automation which only supports Android SDK Platform, API 16 or higher. To support the older APIs, another open source library called Selendroid is used.» [18], работает только с новыми версиями Android.

Appium объединяет в себе оба подхода автоматического тестирования мобильных приложений под Android. Что дает возможность тестировать как старые (<4.2) версии Android посредством Instrumentation подхода, так и системные взаимодействия с помощью UIAutomator. Такая функциональность нам и нужна.

Но не все так гладко с Appium. Как можно увидеть из схемы (Рисунок 3.), Appium использует оба подхода, Instrumentation и UIautomator. Но «честно», они это делаю только в случае с UIAutomator походом. А вот по части Instrumentation подхода, Appium является оберткой над Selendroid, которая переводит команды Selendroid WebDriver в Selendroid команды. В итоге, получаем, что для версий Android 4.2+ используется Google UiAutomator, а для версий Android 2.3+ используется Selendroid, работающий посредством Google Instrumentation.

По сравнению с Selendroid, проект Appium удовлетворяет большему количеству поставленных нами требований.

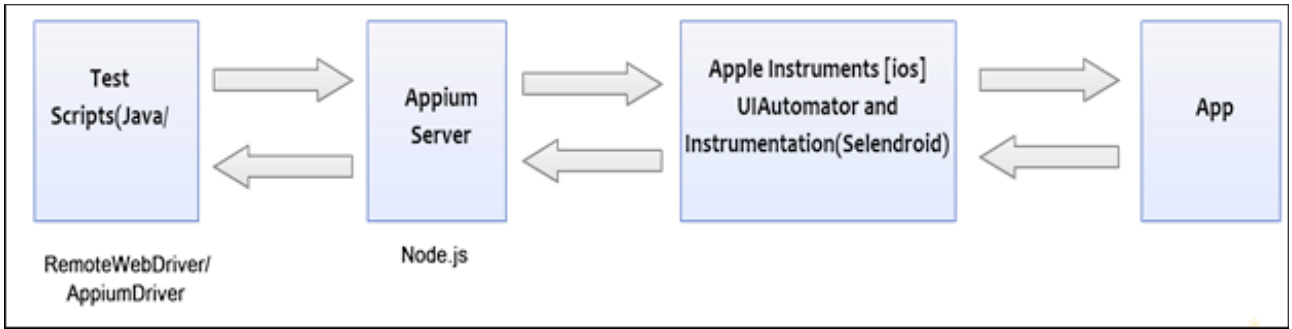


Рис. 3. Схема составных частей Appium [14]

1.2.3. Недостатки готовых решений

Недостатком всех инструментов, использующих Instrumentation подход, в том числе и проекта Selendroid, является полное отсутствие возможности тестирования системных взаимодействий. Автоматическому тестированию подлежит только трестируемое приложение, доступ вне приложения отсутствует.

Подход UIAutomator имеет возможность тестирования системных взаимодействий, но обладает другим недостатком. Он связан с отсутствием возможности работы со старыми версиями Android. Это означает, что все инструменты, использующие только данный подход, тоже будут обладать данным недостатком.

Проект Appium сделал первый шаг к устранению данной проблемы. В нем используются UIAutomator подход, но кода необходимо тестировать приложение на старых версиях Android, Appium использует Selendroid. Один из минусов данного решения состоит в том, что Appium — это не один проект а два отдельных. Appium включает в себя проект сторонних разработчиков — Selendroid, работающий для тестирования версий Android <4.2. Кроме данной проблемы, Apium обладает рядом недостатков [15]:

- между тестами куки не очищаются
- не работает js метод click()
- бывает инспектор не показывает некоторые атрибуты элементов
- инструмент развивается отдельно от сервера.
- инспектор иногда перестает работать
- не находит некоторые элементы
- инспектор может возвращать некорректные локаторы xpath

Наличие приведенных недостатков, делает невозможным использование Appium в качестве надежного инструмента автоматического тестирования.

1.2.4 Итоги

В ходе исследования рынка, на предмет наличия готового решения, было выявлено, что существует ряд инструментов, которые частично удовлетворяют выдвинутым требованиям. Однако, не было найдено решения, обладающего всеми достоинствами одновременно. Потребность в таком решении подвигла на разработку своего проекта.

1.3. Постановка задачи

В данной главе были рассмотрены два основных подхода к автоматизированному тестированию на ОС Android, без изменений трестируемого мобильного приложения, Instrumentation и UIAutomator. Были рассмотрены основные представители обоих подходов: Selendroid и Appium.

Подводя итоги проделанного обзора, хочется отметить, что среди существующих инструментов, нет единого решения позволяющего работать по обоим подходам.

Работа посвящена проектированию и разработке своего решения для автоматического тестирования мобильных приложений под Android. Решение должно удовлетворять выдвинутым требованиям.

Задача — разработать методику автоматизированного тестирования приложений на ОС Android, объединяющую существующие подходы с целью гибкого использования преимуществ каждого из них. Как следствие, — разработать инструмент, объединяющий подходы автоматического тестирования Instrumentation и UIAutomator.

ГЛАВА 2. Теоретическое исследование

В данной главе к рассмотрению предлагается методика тестирования мобильных приложения на ОС Android, предусматривающая возможность тестирования системных взаимодействий и поддерживающая работу с версиями Android <4.2. Рассмотрены теоретические аспекты создания инструмента автоматического тестирования объединяющего подходы Instrumentation и UIAutomator. Так же приведена общая схема инструмента.

2.1. Методика

Для достижения цели данной работы обозначим методику автоматического тестирования Android приложений.

Целью тестирования является выявление и локализация ошибок в работе пользовательского интерфейса и логики приложения.

Этапы тестирования:

- Подготовка к тестированию. Установка тестируемого приложения и сервера тестирования на устройство, анализ графического интерфейса приложения, проверка того, что элементы графического интерфейса приложения доступны для инструмента автоматизированного тестирования.
- Создание автоматизированных тестов для симуляции специфических пользовательских действий над трестируемым приложением.
- Запуск автоматизированного тестирования и анализ полученных результатов.

Принципы тестирования следующие: автоматический тест является клиентом, отправляющим тестовые команды на сервер, где они обрабатываются и пересылаются на другой сервер, работающий на устройстве и взаимодействующий с трестируемым приложением.

Средства, используемые в качестве инструментов для применения данной методики, должны обладать, указанными в данной работе, требованиями.

- Метод работы с версиями Android ОС <4.2 — Instrumentation.
- Метод работы с системными взаимодействиями — UIAutomator.
- Применение методики основано на использовании одного инструмента.
- Трестируемое приложение не должно подвергаться изменениям.

2.2. Предлагаемое решение

Для решения поставленной задачи, предлагается разработать инструмент на базе уже существующего проекта авто тестирования одного из подходов.

Очевидными кандидатами являются проекты Selendroid и Appium. Оба проекта разрабатываются как open-source. Selenium использует Instrumentation. Appium использует UIAutomator подход и проект Selenium. Как упоминалось в обзоре, Appium уже сделал свой шаг в сторону использования подхода Instrumentation. А Selenium, еще не делал подвижек в сторону UIAutomator подхода. Это склоняет наш выбор в сторону Selenium, т.к. способ использования Instrumentation подхода, который избрал Appium, не удовлетворяет нашим требованиям.

Рассмотрим подробнее схему измененного проекта Selendroid (Рисунок 4.) Синим отмечены модификации.

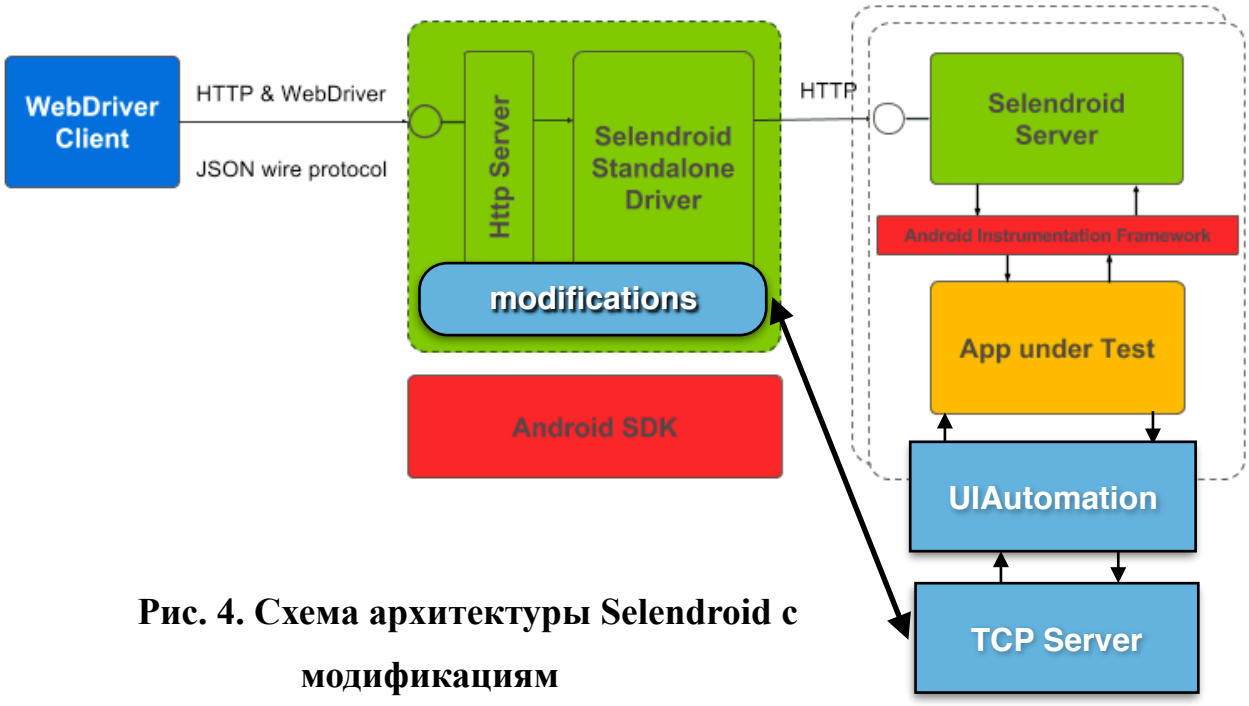


Рис. 4. Схема архитектуры Selendroid с модификациям

В исходной архитектуре Selendroid проекта, модификациям подлежит только Selendroid-standalone. WebDriver Client никак не изменится. Это означает что все тесты, написанные раннее с использованием WebDriver, останутся работоспособными без каких либо изменений. Тестовое приложение также останется без изменения. В системе появится новый элемент — BootstrapServer, выполняющий функции связующего звена между тестовым приложением и модифицированным Selendroid-standalone сервером. BootstrapServer можно использовать из проекта Appium без каких-либо изменений.

2.3. Требования

Предложенное решение удовлетворяет всем требованиям, изложенным ранее. Объединив подходы Instrumentation и UIAutomator, мы получим гибкий инструмент, обладающий полезной функциональностью обоих подходов. В результате создания предложенного решения, мы получим единый инструмент имеющий возможность тестировать системные взаимодействия и работающий со старыми версиями Android.

2.4. Теоретическое сравнение

Существующие методики автоматизированного тестирования основаны на использовании одно из подходов тестирования. Пример такой методики описан в данной работе [19]. Нам важно что эти методики базируются на использовании инструментов работающих по одному из подходов, и как следствие, они имеют ограниченные возможности.

Как уже было подмечено ранее, Appium — проект который уже сделал некоторые попытки объединения возможностей обоих подходов в одном месте. Сделали они это путем написания «обертки» над проектом Selenium. Это дало им возможность тестировать старые версии Android.

Наше решение отличается тем, что у нас будет один инструмент на базе Selendroid со встроенной возможностью переключаться в режим работы с UIAutomator. Selendroid позволяет тестировать более старые версии, а переход в режим UIAutomator позволит тестировать системные взаимодействия. Все это в одном инструменте, а режим работы можно будет переключать в коде самого теста. Такое решение позволяет гибко пользоваться преимуществами обоих подходов, и исключает необходимость использовать два разных инструмента, с разными командами разработки.

ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

В предыдущих главах были описаны концепции положенные в основу системы автоматического тестирования мобильных приложений. В этой главе описаны технические подробности реализации данной системы.

3.1. Интерфейс теста

С точки зрения теста, в нашем инструменте не должно ничего измениться. Тест должен использовать стандартный WebDriver. Возникает вопрос: как сообщить Selendroid серверу, что мы хотим начать работать в режиме UIAutomator.

Было решено использовать следующую технику. В протоколе WebDriver есть метод: session/:sessionId/window. Он используется, чтобы перейти в режим WebView и обратно, в Native режим. Запускается с параметром «WEB» и «NATIVE».

Для перехода в режим UIAutomator, используется этот же метод с параметрами «UIAUTOMATOR» и «INSTRUMENTATION».

```
//переход из режима работы Instrumentation в UIAutpmator
driver.switchTo().window(«UIAUTOMATOR");

// перезод из режима работы UIAutomator в Instrumentation
driver.switchTo().window("INSTRUMENTATION");
```

3.2. Архитектура

В данном раздела будет представлена архитектура реализованного решения. Основным изменениям подвергся Selendroid-standalone сервер, являющийся ядром взятого за основу решения. (Рисунок 5.)

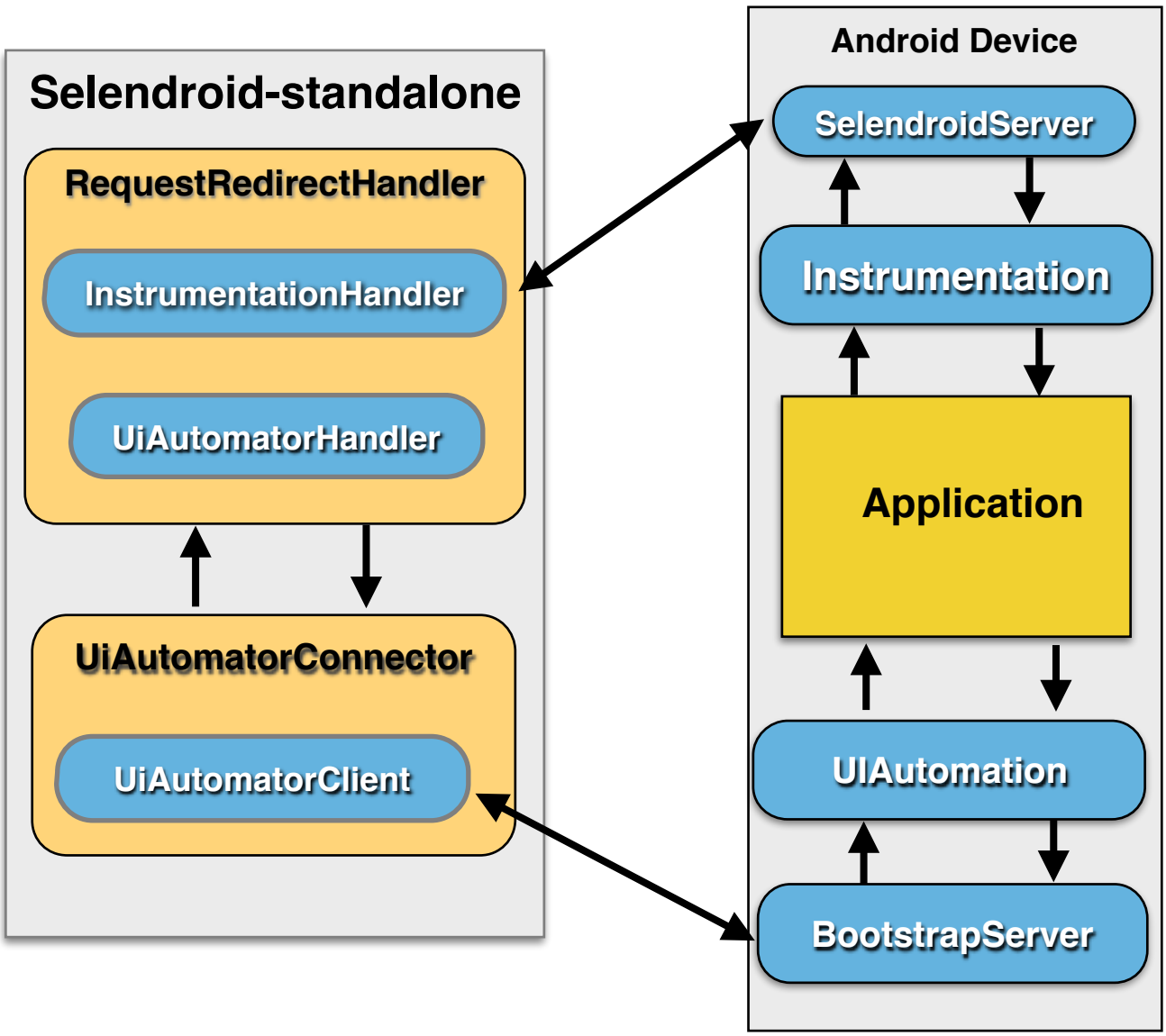
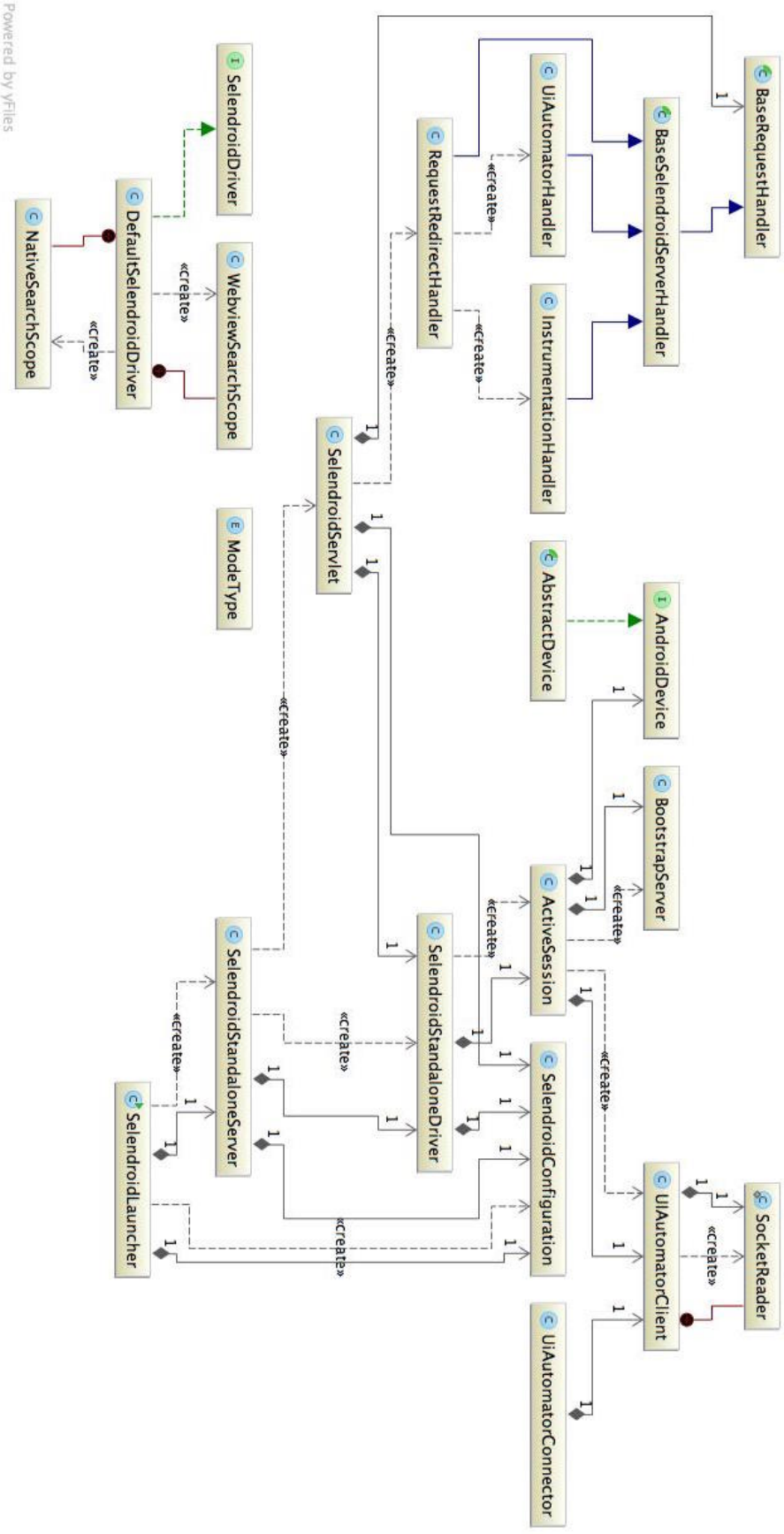


Рис. 5. Архитектура реализованного инструмента


BaseRequestHandler
BaseSelendroidServerHandler
UiAutomatorHandler
RequestRedirectHandler
InstrumentationHandler
SelendroidDriver
DefaultSelendroidDriver
NativeSearchScope
WebviewSearchScope
SelendroidServlet
ModeType
AbstractDevice
AndroidDevice
BootstrapServer
ActiveSession
SelendroidConfiguration
SelendroidStandaloneDriver
SelendroidStandaloneServer
SelendroidLauncher
SocketReader
UiAutomatorClient
UiAutomatorConnector
«create»
Powered by yFiles

Тест является клиентом, взаимодействующим с Selendroid-standalone сервером с помощью JSONE wire протокола. Сам клиент, WebDriver, не изменился. Все запросы, отправляемые клиентом, попадают в RequestRedirectHandler. Далее, поведение Selendroid-standalone сервера зависит от режима, в котором он находится, UIAutomator или Istrumentation. Если сервер в режиме Instrumentation, то запрос передастся в InstumentationHendler, который, в свою очередь, передает запрос Selendroid серверу по HTTP. Selendroid server взаимодействует с трестируемым приложением с помощью Instrumentation API.

Если же, Selendroid-standalone находится в режиме «UIAutomator», то RequestRedirectHandler передаст запрос в UiAutomatorHandler, который передаст запрос в UiAutomatorConnector. UiAutomatorConnector нужен для переработки запроса вида WebDriver в запрос для Bootstrap. Далее, переработанный запрос передается в UiAutomatorClient, который и пересылает его Bootstrap серверу UiAutomatorClient подключенн к Bootstrap серверу, только если Selenium-standalone в режиме UIAutomation. В остальное время он бездействует.

Данная архитектура разрабатывался с возможностью параллельного запуска тестов. Это возможно благодаря UiAutomatorClient. Он привязан к тестовой сессии. Таких сессий можно создавать много. На каждую будет выделен девайс или эмулятор. Состояния сессий независимы. Одна может находиться в Instrumentation режиме, а другая в UIAutomator

На рисунке 6 представлена UML диаграмма окончательной реализации разработанного инструмента.

Рис. 6. UML диаграмма реализованного инструмента

Пример команды запуска инструмента:

java -jar selendroid-standalone-with-uiautomator-mode.jar -aut ok.apk

Приведем пример теста с использованием разработанного инструмента (листинг 1):

Листинг 1 Тест работы приложения с отключением интернет соединения

```
import io.selendroid.SelendroidCapabilities;
import io.selendroid.SelendroidDriver;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.touch.TouchActions;
import org.openqa.selenium.remote.DesiredCapabilities;

public class ExampleTest {
private SelendroidDriver driver = null;
   @Before
   public void startSelendroidServer() throws Exception {
      DesiredCapabilities caps = SelendroidCapabilities.android();
      driver = new SelendroidDriver(caps);
   }
   @Test
   public void notificationTest() throws Exception {
      // Изначально находимся в Instrumentation режиме
      // Имеем доступ только к объектам из трестируемого приложения
      driver.get("http://m.ok.com");
      int height = driver.findElement(By.xpath(«/*")).getSize().height;
      // Переключаемся в режим UIAutomator
```

```
    driver.switchTo().window(«UIAUTOMATOR");

    // Теперь есть доступ и к элементам вне приложения.
    // Оттягиваем «шторку» уведомлений
    new TouchActions(driver).down(25, 25).move(25, height).
            up(25, height).perform();
   // Находим кнопку wifi
    WebElement wifiButton = driver.findElement(
            By.xpath(«//*[contains(@text,'WiredSSID')]"));
    // Переходим в настройки wifi
    wifiButton.click();
    // Находим выключатель wifi
    WebElement wifiSwitch = driver.findElement(
            By.xpath(«//*[contains(@text,'On')]"));
    // Выключаем wifi
    wifiSwitch.click();
    // Вызываем нажатие физической кнопки back на устройстве
    // Возвращаемся в трестируемое приложение
    driver.hardBack();
    // Закончили необходимые действия вне прилжения
    // Переходим в Instrumentation режим
    driver.switchTo().window(«INSTRUMENTATION»);
    WebElement connectionMessage = driver.findElement(
            By.xpath(«//*[contains(@text,'connection')]"));
    AssertEquals(«На главном экране отобразилось сообщение о разрыве
      интернет соединения.», «No internet connection!»,
      connectionMessage.getText());
}
@After
public void teardown() {
    // Завершаем тестовую сессию
    driver.quit();
}
```

```
}
```

3.2. Практическое применение

Разработанное решение было внедрено в реальный проект автоматического тестирования мобильных приложений на предприятии. Ранее, для тестирования системных взаимодействий, использовались тесты, напрямую работающие с UIAutomator. Такие тесты, в виде jar файлов, копировались на тестовое устройство и запускались через adb shell обычным Selendroid тестом [17]. Статус выполнения таких тестов недоступен. Для изменения данного теста, его требуется переписать, скомпилировать, и снова записать на устройство. Разобраться, в чем причина падения такого теста, не представляется возможным. Данные тесты трудны в разработке и поддержании их в рабочем состоянии. По этой причине, количество тестов системных взаимодействий трестируемого приложения было критически мало.

Разработанное решение сразу позволило избавиться от ранее используемых тестов, напрямую работающих с UIAutomator. В настоящее время, данная разработка обеспечивает функционирование более тридцати автоматизированных тестов взаимодействующих с системой Android. Это существенно освободило время работников ручного тестирования для проведения более интеллектуального тестирования, что привело к повышению качества конечного продукта. Кроме того, учитывая возможности разработанного инструмента, было принято решение о покрытии автоматическими тестами около семидесяти новых тестовых сценариев , основанных на взаимодействии с системой. Это покроет большую честь наиболее важных системных взаимодействий.

Тестированию подлежат все 34 вида push-уведомлений (Рисунок 7). Перечислим некоторые из них:

- приглашение в группу

- новое сообщение
- новый комментарий
- видео звонок
- приглашение в друзья
- поставили класс и т.д.

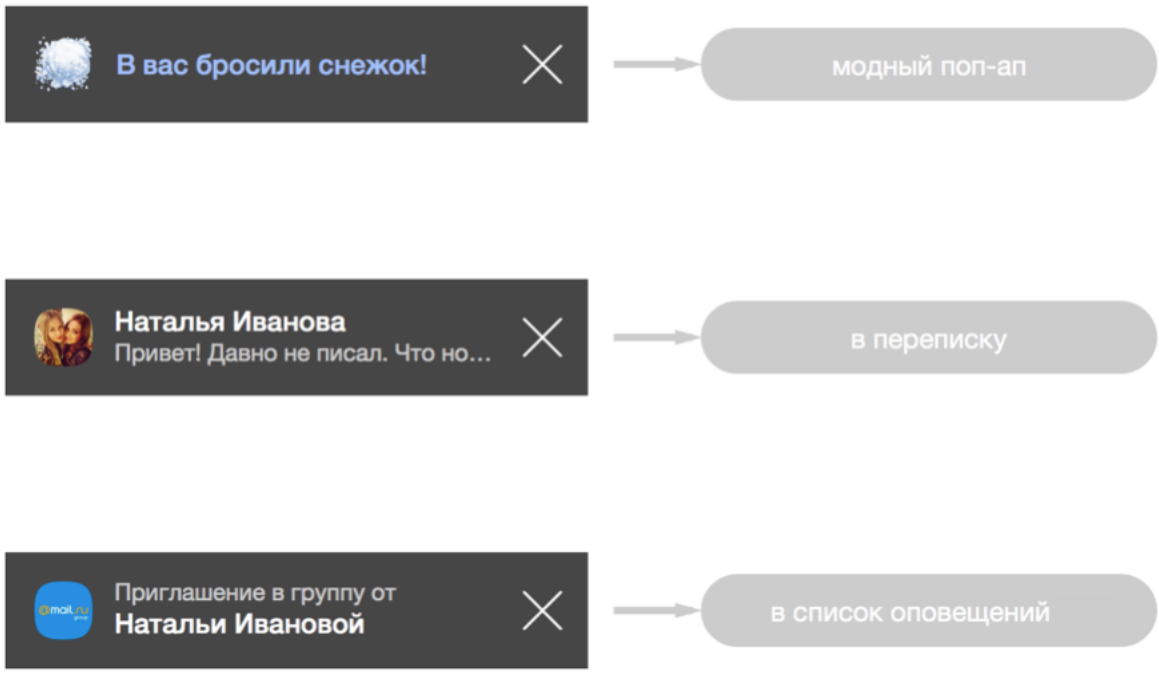


Рис. 7. Push-уведомления и разделы в которые они ведут

Так же возможны разные состояния системы: телефон заблокирован, приложение закрыто, открыто, в фоне, в настройках приложения включены уведомления со звуком, без звука, в настройках приложения включены полные уведомления, краткие.

Для подсчета примерного числа возможных состояний системы, необходимо 34 вида push-уведомления умножить на 8 состояний системы, что составляет более 270-ти тестовых сценариев . Было решено выделить 100 наиболее важных тестовых сценариев, 30 из которых был реализованы в рамках тестирования работоспособности разработанного решения в условиях проекта контроля качества продукта на производстве. Решение показало себя надежным, легким во внедрении и удобным в использовании.

Заключение

В рамках представленной в данной работе методики, был разработан инструмент автоматизированного тестирования приложений под ОС Android на основе подходов Instrumentation и UIAutomation. Инструмент был успешно опробован и протестирован в условиях работы отдела качества продукта, на предприятии.

Успешно пройдя тестирование, инструмент был внедрен в проект автоматического тестирования мобильных приложений на предприятии.

Использование инструмента дало возможность не только сильно улучшить качество существующих автоматизированных тестов системных взаимодействий, но и написать новые тесты системны взаимодействий.

На данный момент инструмент успешно используется в составе системы автоматического тестирования на предприятии.

ИСТОЧНИКИ

[1] Дастин Э., Рэшка Дж., Пол Дж. Автоматизированное тестирование программного обеспечения. /Э. Дастин, Дж. Рэшка, Дж. Пол. – Москва: ЛОРИ, 2003.

[2] Burns David. Selenium 2 Testing Tools: Beginner's Guide // Paperback use pre formatted date that complies with legal requirement from media matrix – October 19, 2012.

[3] Информационный сайт, посвященный автоматизации тестирования. URL: http://automated-testing.info/.

[4] Test automation for native or hybrid Android apps and the mobile web with Selendroid. URL: http://selendroid.io/.

[5] Selenium / WebDriver. URL: http://selenium2.ru/.

[6] Notifications. URL: https://developer.android.com/guide/topics/ui/notifiers/notifications.html/.

[7] Instrumentation Class Overview. URL: http://developer.android.com/reference/android/app/Instrumentation.html/.

[8] Testing Fundamentals. URL: http://developer.android.com/tools/testing/testing_android.html/.

[9] Testing Support Library. URL: https://developer.android.com/tools/testing-support-library/index.html/.

[10] Ebay inc. URL: http://www.ebayinc.com/.

[11] SeleniumHQ URL: http://www.seleniumhq.org/projects/webdriver/.

[12] Selenium Grid. URL: http://www.seleniumhq.org/projects/grid/.

[13] Appium automation for App. URL: http://appium.io/.

[14] Valsatech Mobile Test Automation. URL: http://www.valsatech.com/mobiletest.html/.

[15] Appium в серьёзных задачах URL: http://habrahabr.ru/company/yandex/blog/227571/.

[16] Platform Versions. URL: https://developer.android.com/about/dashboards/index.html/.

[17] Автоматизация тестирования Android приложений с помощью UIAutomator. URL: http://habrahabr.ru/company/intel/blog/205864/.

[18] Shah G., Shah P., Muchhala R. Software testing automation using appium. – 2014. URL: http://inpressco.com/wp-content/uploads/2014/10/Paper793528-3531.pdf/.

[19] Дерев’янченко О. В., Плакидюк В. О. Система автоматичного тестування програм під ОС Android //Искусственный интеллект. – 2013. – №. 4. – С. 442-453.

Приложение

Листинг теста Push-уведомления.

```
@Test
 public void notificationTest() {
    user1 = BotManager.getRandomUser();
    user2 = BotManager.getRandomUser();
    apiUser1 = new Messages(user1);
    apiUser2 = new Messages(user2);

    login(user1);
    String textInNotification =  MessagesUtils.generateRandomText();
    apiUser2.sendMessageTo(user1.getId(), textInNotification);
    Controls.scrollDownFullScreen(driverId);
    Webview.switchToUIAutomatorMode(driverId);
    WebElement element = driver.findElement(By.xpath(locator));
    String notificText = element.getText();
    element.click();
    Webview.returnFromUIAutomatorMode(driverId);

    assertTrue("Message in notifications is incorrect. Expected: " + user2
       + ": " + textInNotification + ", actual: " + notificText,
              notificText.equals(user2 + ": " + textInNotification), logger);

    assertTrue("Last message in chat is incorrect. Expected: " +
       textInNotification + ", actual: " + chatPage.getLastMessageText(),
       chatPage.getTheLastOfTheLastMessageText().equals(
       textInNotification), logger);
    logger.testWorkedFine();
 }
```