МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой №1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., док. физмат. наук, зав. каф |  |  |  | Смирнов А.О. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

|  |  |
| --- | --- |
| на тему | Применение средств автоматизации для повышения эффективности |
| тестирования мобильных приложений | |
|  | |

|  |  |
| --- | --- |
| выполнена | Барановым Семеном Алексеевичем |
| фамилия, имя, отчество студента в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 01.04.02 |  | Прикладная математика и информатика |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности |  |  |  |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы № | М210М |  |  |  | С.А.Баранов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., док. физмат. наук, зав. каф |  |  |  | Смирнов А.О. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой №1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., док. физмат. наук, зав. каф |  |  |  | Смирнов А.О. |
| должность, уч. Степень, звание |  | подпись, дата |  | инициалы, фамилия |

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

|  |  |  |  |
| --- | --- | --- | --- |
| студенту группы № | М210М |  | Баранов Семен Алексеевич |
|  |  |  | (фамилия, имя, отчество) |

|  |  |
| --- | --- |
| на тему | Применение средств автоматизации для повышения эффективности |
| тестирования мобильных приложений | |

|  |  |  |  |
| --- | --- | --- | --- |
| утвержденную приказом ГУАП от |  | № |  |

|  |  |
| --- | --- |
| Цель исследования: | Определить, насколько эффективнее тестирование с применением |
| средств автоматизации по сравнению с ручным тестированием | |
|  | |

|  |  |
| --- | --- |
| Задачи исследования: | 1. Изучить теоретические основы автоматизации тестирования |
| 1. Проанализировать доступные средства по автоматизации тестирования 3. Создать мобиль- | |
| ное приложение 4. Протестировать полученное приложение с помощью средств автоматизации | |

|  |  |
| --- | --- |
| Содержание диссертации (основные разделы): | 1. Теоретические основы автомати- |
| зации тестирования 2. Анализ доступных средств автоматизации тестирования мобильных | |
| приложений. 3. Создание мобильного приложения 4. Создание и прогон автоматизирован- | |
| ных тестов | |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Срок сдачи диссертации « |  | » |  | 20 |  |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., док. физмат. наук, зав. каф |  |  |  | Смирнов А.О. |
| должность, уч. Степень, звание |  | подпись, дата |  | инициалы, фамилия |

Задание принял к исполнению

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| студент группы № | М210М |  |  |  | Смирнов А.О. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

**ОГЛАВЛЕНИЕ**

**ВВЕДЕНИЕ1**

**1** **Теоретические основы автоматизации тестирования2**

1.1 Роль тестирования в жизненном цикле разработки программного обеспечения3

1.2 Виды тестирования программного обеспечения4

1.3 Ручное тестирование программного обеспечения 5

1.4 Автоматизированное тестирование программного обеспечения 6

**2** **Анализ доступных средств автоматизации тестирования мобильных приложений7**

2.1 Драйверы8

2.2 Надстройки9

**3 Создание мобильного приложения**1**5**

3.1 Определение требований16

3.2 Разработка дизайна приложения17

3.3 Программирование логики18

**4 Создание автоматизированных тестов20**

4.1 Создание тест-кейсов для тестирования21

4.2 Автоматизация тест-кейсов22

4.3 Прогон полученных тест-кейсов и анализ полученных результатов23

# ВВЕДЕНИЕ

**Актуальность исследования** обусловлена широким распространением высоконагруженных мобильных приложений. Современное мобильное приложение имеет обширный функционал, сложную структуру интерфейса, множество интеграций со сторонними сервисами.

Для тестирования таких сложных систем может потребоваться большое количество ресурсов. Недостаточное тестирование может привести к большому количеству ошибок, сбоев, уязвимостям, что в свою очередь может привести к потерям прибыли, а в некоторых специфических сферах к причинению вреда здоровью и даже смертям.

Ручное тестирование подобных систем может занимать значительное время. Внедрение средств автоматизации в процесс тестирования поможет сократить время на тестирование, увеличить его эффективность. Благодаря этому продукты станут более качественными, надежными и безопасными.

**Целью работы** является определить, насколько эффективнее тестирование с применением средств автоматизации по сравнению с ручным тестированием (с учетом времени на разработку и поддержку автоматизированных тестов, стоимости трудозатрат ручного и автоматизированного тестирования).

**Задачи** необходимые для достижения целей данной работы:

1. Изучить теоретические основы автоматизации тестирования;
2. Проанализировать доступные средства по автоматизации тестирования;
3. Создать мобильное приложение, которое необходимо протестировать;
4. Протестировать полученное приложение с помощью средств автоматизации. Оценить эффективность тестирования.

**Научная новизна** обусловлена отсутствием научных исследований в области, которые бы ответили бы на вопросы:

* насколько тестирование с использованием средств автоматизации быстрее, чем тестирование без применения средств автоматизации;
* насколько тестирование с использованием средств автоматизации дешевле, чем тестирование без применения средств автоматизации.

Большинство научных работ в данной предметной области рассказывают о плюсах и минусах, приведенных выше видов тестирования, о проблемах внедрения автоматизированного тестирования в проект, о принципах раннего тестирования, о рекомендациях что именно стоит автоматизировать, но в них не приведены какие-то конкретные числа и примеры.

**Практическая значимость** работы состоит в сокращении времени тестирования за счет принятия решения об внедрения средств автоматизации в процесс разработки мобильного приложения.

**Методы исследования**:

* Теоретический анализ технической документации;
* Сравнение;
* Индукция.

# 1 Теоретические основы автоматизации тестирования

На данный момент автоматизация тестирования в различных его формах внедряется и развивается в широком множестве сложных систем. Его применение в одних случаях помогает увеличить эффективность тестирования, в других случаях просто необходимо ввиду либо невозможности провести ручное тестирование, либо высокой стоимости, сложности его проведения.

Автоматизированные тесты используют такие известные компании как Сбербанк, Альфа-Банк, Google, Amazon, Microsoft и т.д. Для того, чтобы понять нужно ли использовать автоматизацию для конкретного продукта, компании проводят внутренние исследования. На основании этих исследований принимается решение о внедрении автоматизации в процесс тестирования.

Для того, чтобы понять основы автоматизации тестирования, необходимо понять суть мануального (ручного) тестирования. И в целом определить роль тестирования в жизненном цикле разработки программного обеспечения. Любой специалист по UI автотестам или Unit-тестам понимает принципы ручного тестирования.

Перед переходом к рассмотрению тестирования как этапа жизненного цикла разработки программного обеспечения хотелось бы обозначить семь основных принципов тестирования. Эти принципы являются основополагающими для любого вида тестирования.

1. Тестирование демонстрирует наличие дефектов (а не их отсутствие). Тестирование может показать наличие дефектов в объекте тестирования, но не может доказать их отсутствие. Тестирование снижает вероятность того, что дефекты в объекте тестирования останутся необнаруженными, но даже если дефекты не были обнаружены, тестирование не доказывает корректности объекта тестирования.

2. Исчерпывающее тестирование невозможно. Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев. Вместо того, чтобы пытаться провести исчерпывающее тестирование, следует использовать методы тестирования, расстановку приоритетов тестовых сценариев и тестирование, основанное на рисках чтобы сосредоточить усилия по тестированию.

3. Раннее тестирование экономит время и деньги. Дефекты, устраненные на ранней стадии процесса, не вызовут последующих дефектов в производных рабочих продуктах. Стоимость качества будет снижена, так как позже в ЖЦ ПО будет происходить меньше отказов. Для раннего обнаружения дефектов как можно раньше следует начинать как статическое тестирование, так и динамическое тестирование.

4. Кластеризация дефектов. Обычно небольшое количество системных компонентов содержит большинство обнаруженных дефектов или порождает большинство эксплуатационных отказов. Это явление является иллюстрацией принципа Парето. Предсказанные и фактические кластеры дефектов, наблюдаемые в ходе тестирования или эксплуатации, являются важными входными данными для тестирования, основанного на рисках.

5. Тесты устаревают. Если одни и те же тесты повторяются много раз, они становятся все более неэффективными в обнаружении новых дефектов. Для обнаружения новых дефектов может потребоваться изменение существующих тестов и тестовых данных, а также написание новых тестов. Однако в некоторых случаях повторение одних и тех же тестов может иметь положительный результат, например, при автоматизированном регрессионном тестировании.

6. Тестирование зависит от контекста. Не существует единого универсального подхода к тестированию. Тестирование выполняется по-разному в зависимости от контекста.

7. Заблуждение об отсутствии дефектов. Было бы ошибкой ожидать, что верификация программного обеспечения обеспечит успех системы. Тщательное тестирование всех указанных требований и исправление всех обнаруженных дефектов может привести к созданию системы, которая не будет соответствовать потребностям и ожиданиям пользователей, не будет помогать в достижении бизнес-целей заказчика и будет уступать другим конкурирующим системам. Также в дополнение к верификации следует проводить валидацию. (скопировано из ISTQB надо адаптировать под свои цели)

## 1.1 Роль тестирования в жизненном цикле разработки программного обеспечения

Тестирование программного обеспечения имеет большое множество определений, раскрывающих понятие с разных сторон. Тестирование программного обеспечения рассматривают как отдельный этап жизненного цикла программного обеспечения, как процесс, происходящий на всех этапах жизненного цикла ПО, как отдельную дисциплину и т. д.

Ввиду того, что необходимо оценить эффективность автоматизации тестирования по сравнению с ручным тестированием, то наилучшим образом подходит следующее определение:

**Тестирование программного обеспечения** — проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, для получения информации о ее качестве для заинтересованных лиц.[[1]](#footnote-1)

Суть автоматизации тестирования в конечном итоге сравнить ожидаемое поведение программы с ее реальным и, если они не совпали, сообщить об этом.

Сами же понятия ручного и автоматизированного тестирование обычно включают в себя гораздо больше. Здесь же все зависит от обозначенного круга ответственности специалистов по тестированию конкретного продукта. Поэтому, чтобы сравнить эти два вида было необходимо рассмотреть их с одной общей для этих видов стороны.

Любой специалист по тестированию тем или иным образом ищет несоответствия между реальным поведением программы и ее ожидаемым поведением, а затем оформляет их как баг-репорт. Эти баг-репорты являются формой информации о состоянии программного обеспечения.

В данном случае тестирование проявляется себя наибольшим образом как один из этапов жизненного цикла разработки программного обеспечения. Далее будет дано наиболее подходящее определение жизненного цикла программного обеспечения.

**Жизненный цикл разработки программного обеспечения** представляет собой процесс создания программного обеспечения от начала до конца. SDLC включает в себя последовательность фаз, этапов и действий, которые разработчики и инженеры программного обеспечения выполняют для создания и поддержки программного продукта. Общепринятая модель SDLC включает следующие основные этапы: [[2]](#footnote-2)

1. **Планирование. На этой стадии определяются цели проекта, ресурсы, риски, бюджет.**
2. **Анализ требований**. В этой фазе происходит сбор и анализ требований пользователей и бизнеса. Определяются функциональные и нефункциональные требования к программному продукту.
3. **Проектирование. На этом этапе происходит разработка архитектуры программного продукта, определяются технологии, инструменты и платформы разработки.**
4. **Реализация (разработка). В этой фазе программисты начинают писать код, реализуя функциональность и логику программного продукта на выбранном языке программирования. Стоит отметить, что на данном этапе пишутся автотесты, такие как UI-тесты, Unit-тесты.**
5. **Тестирование. После завершения разработки проводятся тесты для проверки работоспособности, безопасности и соответствия требованиям программного продукта.**
6. **Внедрение (развертывание). После успешного завершения тестирования программное обеспечение разворачивается на целевой среде и становится доступным для конечных пользователей. (Это может быть общедоступный домен сайта, либо публикация приложения на любой общедоступной площадке такой как AppStore, Google Play, RuStore, App Gallery и другие, так же встречается распространение любым другим образом)**
7. **Сопровождение (поддержка). После внедрения происходит постоянная поддержка и обновление программного продукта, включая исправление ошибок, добавление новых функций и адаптацию к изменениям в окружающей среде.**[[3]](#footnote-3)

Важно отметить, что тестирование должно быть «сквозным», то есть проводиться на всех этапах жизненного цикла программного обеспечения тем или иным образом. Чем раньше будет внедряться тестирование в проект, тем лучше.

Например:

* На этапе **анализа требований** документация проверяется на завершенность, недвусмысленность и согласованность. Главная задача — убедиться в том, что требования были правильно интерпретированы и остаются корректными, понятными и последовательными.
* На этапе **разработки важно** проводить юнит-тестирование, интеграционное тестирование и системное тестирование. В самом начале этого этапа разработки проводится юнит тестирование. Этот процесс представляет собой проверку отдельного модуля системы или функционала. Интеграционное тестирование проводится после того, как несколько модулей объединены вместе как отдельная часть приложения. В дальнейшем в процессе разработки все больше и больше модулей объединяются воедино. После того, как разработка закончена, наступает время подготовки к системному тестированию.
* На этапе **тестирования** тесты проводятся независимо от того, проводились ли они на предыдущих этапах. Должно быть проведено полное функциональное тестирование и тестирование пользовательских интерфейсов, а все обнаруженные дефекты должны быть задокументированы в системе баг-трекинга. Помимо этого, применяется регрессионное тестирование. После завершения отладки предоставляется оценка общего качества продукта. После завершения последнего теста процесс тестирования программного обеспечения считается законченным. *На этом этапе так же пишут и исполняют авто-тесты.*
* На этапе **сопровождения** остается необходимость в тестировании, проводимом на этапе эксплуатации и поддержки. Разные пользователи могут работать в абсолютно разных окружениях. Поэтому всегда возможно, что новые ошибки, которые не были выявлены ранее дадут о себе знать. Более того, пользователи могут использовать программное обеспечение изначально непредвиденным способом. Это, в свою очередь, может вызвать некоторые непредвиденные проблемы. В таком случае потребуется вмешательство отдела обеспечения качества.

Выводы:

Таким образом, тестирование, с одной стороны, является одним из этапов жизненного цикла разработки программного обеспечения, включающего планирование, анализ требований, проектирование, реализацию, тестирование, внедрение, сопровождение и поддержку.

С другой стороны, тестирование пронизывает каждый из вышеперечисленных этапов. Для повышения качества и соответствия требованиям продукта тестирование в той или иной форме должно проводиться на каждом из этапов жизненного цикла программного обеспечения.

Для удобства далее тестирование будет рассматриваться как один из этапов жизненного цикла. При чем, тестирование будет рассмотрено как сравнение реального поведения программы с ее ожидаемым поведением на ограниченном наборе тестов. Только таким образом, возможно сравнить два его вида – автоматизацию тестирования и ручное тестирование.

## Виды тестирования программного обеспечения

В контексте тестирования программного обеспечения существует большое множество видов тестирования. Для некоторых видов тестирования существуют отдельные профессии. Например, тестирование безопасности, нагрузочное тестирование, автоматизация тестирования, модульное тестирование (unit-тесты).

Далее будут рассмотрены лишь небольшое количество видов, достаточных для достижения целей работы и выполнения поставленных задач.

**По подходам к исполнению тестирования выделяют:**

* Скриптовое (сценарное) тестирование – это формализованный подход к тестированию, основанный на использовании заранее подготовленных тест-кейсов и наборов тест-кейсов. Это самый распространённый способ тестирования, позволяющий достичь максимальной полноты исследования продукта благодаря строгой систематизации процесса, удобству применения метрик и широкому набору рекомендаций.
* Исследовательское тестирование - частично формализованный подход, в рамках которого тестировщик работает с продуктом по выбранному сценарию, который может дорабатываться в процессе выполнения для более полного исследования приложения.
* Сессионное тестирование - компромисс между исследовательским и скриптовым подходом. Процесс тестирования разбивается на сессии определенной длины (Обычно сессия длится не более 90 минут).

**По степени автоматизации:**

* Ручное тестирование — это процесс проверки программного обеспечения без использования автоматизированных инструментов. Тестировщик взаимодействует с программой как обычный пользователь, выполняя различные сценарии использования и тестовые сценарии, вводя данные, наблюдая за результатами и проверяя наличие ошибок или неожиданного поведения.
* Автоматизированное тестирование — это процесс проверки программного обеспечения с использованием специальных программных инструментов, которые выполняют тесты автоматически, без участия человека. Тестировщик создаёт скрипты или сценарии тестирования, содержащие инструкции для выполнения определённых действий и проверки результатов. Эти сценарии запускаются на специальных инструментах для автоматизации тестирования, которые эмулируют действия пользователя и анализируют результаты выполнения.
* Полуавтоматизированное тестирование — это процесс, при котором часть тестов выполняется автоматически с помощью автоматизированных инструментов, а другая часть — вручную тестировщиками. Такой подход является наиболее распространенным, так как в некоторых случаях автоматизировать сценарий бывает сложно, невыгодно либо просто невозможно.

Так же к последнему подходу относится применение различных средств, например, регулярных выражении (Regex), средств мониторинга трафика (Charles, Fiddler, Chrome Dev Tools), средств автоматизации вызовов и анализа HTTP-запросов (Postman).

Это позволяет сочетать преимущества ручного и автоматизированного тестирования, обеспечивая более полный и качественный анализ программного обеспечения.

**По принципам работы с приложениями:**

* Позитивное — это тестирование с использованием сценариев, соответствующих нормальному поведению системы. Оно помогает определить, что система выполняет свои функции должным образом.
* Негативное тестирование, наоборот, использует сценарии, соответствующие нештатному поведению системы. Оно проверяет устойчивость системы к различным воздействиям, правильности обработки ошибок и работу с некорректными данными.

**По запуску кода на исполнение:**

* Статическое тестирование — это процесс проверки программного обеспечения без запуска кода на исполнение. Оно включает обзоры, анализ кода и статический анализ. Статическое тестирование проводится на ранних этапах жизненного цикла программного обеспечения и помогает обнаружить дефекты в документах и коде.
* Динамическое тестирование — это процесс проверки программного обеспечения путём запуска кода на исполнение. Динамическое тестирование проводится на более поздних этапах жизненного цикла программного обеспечения и помогает проверить реальное поведение и функциональность приложения.

**По целям и задачам:**

* Функциональное тестирование — это проверка способности программного обеспечения решать задачи, необходимые пользователям. Оно включает тестирование функциональной пригодности, точности, способности к взаимодействию, соответствия стандартам и правилам, а также защищённости.
* Нефункциональное тестирование направлено на проверку реализуемости нефункциональных требований, таких как производительность, удобство использования, портируемость (установка) и надёжность (отказ/восстановление), возможность обновления, безопасности.

Нефункциональный вид тестирования в свою очередь делится на несколько подвидов:

* Тестирование производительности — это вид тестирования, который проводится для определения скорости или эффективности работы системы под нагрузкой. Оно включает в себя определение скорости обработки, надежности и масштабируемости системы.
* Тестирование совместимости — это процесс проверки, как ваше программное обеспечение будет работать в различных средах, включая различные операционные системы, браузеры, сетевые окружения и т.д. Целью является убедиться, что ваше ПО совместимо со всеми потенциальными рабочими средами.
* UX-тестирование (тестирование удобства пользования) — это процесс, в ходе которого определяется, насколько удобно и просто пользователю взаимодействовать с продуктом. Это включает в себя понимание, насколько интуитивно понятен интерфейс, как легко осуществляется навигация, и насколько удовлетворены пользователи в целом.
* Тестирование надежности — это вид тестирования, который проводится для определения способности системы или компонента выполнять требуемые функции без сбоев в определенных условиях и в течение определенного периода времени.
* Тестирование безопасности — это процесс идентификации уязвимостей и слабых мест в системе с целью предотвращения несанкционированного доступа, потери данных, вторжений и других угроз.
* Тестирование портируемости — это процесс проверки, насколько легко программное обеспечение может быть перенесено с одной среды в другую. Это может включать в себя перенос между различными операционными системами, различными версиями одной и той же операционной системы или даже различными оборудованиями.

**По доступу к коду и архитектуре программного обеспечения:**

* Тестирование белого ящика — это процесс проверки кода и внутренней структуры программного обеспечения.
* Тестирование чёрного ящика — это поведенческое тестирование, при котором специалист по тестированию не знает внутреннюю структуру и устройство системы.
* Тестирование серого ящика — это комбинация тестирования белого и чёрного ящиков, где специалист имеет частичное представление о внутренних процессах системы.

**По уровню тестирования:**

* Модульное тестирование — это тестирование отдельных модулей или компонентов программы, например, функций или классов. Цель модульного тестирования — выявление ошибок в реализации алгоритмов и определение степени готовности системы к разработке и тестированию.
* Интеграционное тестирование — это тестирование взаимодействия между несколькими модулями системы. Основная задача интеграционного тестирования — поиск дефектов, связанных с ошибками в интерфейсном взаимодействии между модулями.
* Системное тестирование рассматривает тестируемую систему в целом и оперирует на уровне пользовательских интерфейсов. Основная задача системного тестирования — выявление дефектов, связанных с работой системы в целом, таких как неправильное использование ресурсов, несовместимость с окружением и неудобство в применении.
* Приемочное тестирование фокусируется на проверки и демонстрации готовности системы к разворачиванию, что означает удовлетворение системой пользовательских потребностей. Основными формами приемочного тестирования являются: пользовательское приемочное тестирование, эксплуатационное приемочное тестирование, контрактное и нормативное приемочное тестирование, альфа-тестирование и бета-тестирование.

**По природе программного обеспечения:**

* Тестирование веб-приложений
* Тестирование настольных приложений
* Тестирование мобильных приложений

Эти виды тестирования отличаются концентрацией на определённых функциях и особенностях приложений, использованием специфических инструментов и техник.

Выводы:

В тестировании существует множество разнообразных видов тестирования программного обеспечения, каждый из которых имеет свои особенности и применяется в зависимости от преследуемых целей и поставленных задач.

Виды тестирования программного обеспечения классифицируется по:

1. По природе программного обеспечения;
2. По уровню тестирования;
3. По доступу к коду и архитектуре программного обеспечения;
4. По целям и задачам;
5. По запуску кода на исполнение;
6. По принципам работы с приложениями;
7. По степени автоматизации;
8. По подходам к исполнению тестирования.

Таким образом, разнообразие видов тестирования и подходов позволяет выбирать наиболее подходящий способ и направление для обеспечения качества программного обеспечения в соответствии с требованиями и целями проекта.

## Ручное тестирование программного обеспечения

Ручное тестирование — это процесс проверки программного обеспечения, выполняемый вручную тестировщиками без использования автоматизированных инструментов или скриптов. В ходе ручного тестирования тестировщики взаимодействуют с приложением, проверяют его функциональность, интерфейс, производительность и другие аспекты, чтобы выявить ошибки и дефекты.

Цели и задачи ручного тестирования мобильных приложений включают:

* Выявление ошибок и дефектов: Основная цель ручного тестирования - обнаружить ошибки и дефекты в мобильном приложении;
* Проверка функциональности: Ручное тестирование позволяет проверить, работает ли приложение согласно заданным требованиям и функциональности;
* Оценка пользовательского опыта: Ручное тестирование позволяет оценить удобство использования приложения и обнаружить проблемы с интерфейсом или взаимодействием с пользователем;
* Тестирование различных сценариев: Ручное тестирование позволяет проверить приложение на различные сценарии использования, включая краевые случаи и нестандартные ситуации.

Ограничения ручного тестирования мобильных приложений включают:

* Ограниченные ресурсы: Ручное тестирование может быть ограничено доступными ресурсами, такими как время и количество тестировщиков.
* Сложность повторяемости: Ручное тестирование может быть менее повторяемым, поскольку результаты могут зависеть от индивидуальных навыков и опыта тестировщика. *Все люди разные, даже специалисты по ручному тестированию, для проверки одного и того же сценария может потребоваться разное количество времени, а также могут отличаться результаты одной и той же проверки.*
* Ограниченная покрытие: Ручное тестирование может не обеспечить полное покрытие всех возможных сценариев использования приложения. *Этой является следствием ограниченности временных и кадровых ресурсов.*
* Зависимость от тестировщиков: Ручное тестирование требует наличия опытных и квалифицированных тестировщиков для достижения хороших результатов. *Один и тот же тестовый сценарий может обнаружить дефект либо не обнаружить в зависимости от квалификации и опыта тестировщика.*

При помощи ручного тестирования возможно обнаружить различные дефекты в мобильных приложениях, такие как:

* Ошибки функциональности: Ручное тестирование позволяет проверить, работает ли приложение согласно заданным требованиям и функциональности.
* Проблемы с пользовательским интерфейсом: Ручное тестирование позволяет обнаружить проблемы с интерфейсом приложения, такие как неправильное отображение элементов или неудобство использования.
* Проблемы с производительностью: Ручное тестирование может помочь выявить проблемы с производительностью приложения, такие как медленная загрузка или задержки в ответе на действия пользователя.
* Проблемы совместимости: Ручное тестирование позволяет проверить, как приложение работает на разных устройствах, операционных системах и разрешениях экрана.
* Проблемы безопасности: Ручное тестирование может помочь выявить уязвимости и проблемы безопасности в приложении.

Однако, есть некоторые дефекты, которые могут быть сложно обнаружить при помощи ручного тестирования. Например:

* Сложные алгоритмы и логика: Ручное тестирование может быть ограничено в выявлении сложных ошибок в алгоритмах и логике приложения.
* Масштабируемость и нагрузочное тестирование: Ручное тестирование может быть ограничено в проверке производительности и масштабируемости приложения при больших нагрузках.

Вывод:

Ручное тестирование можно идеально подходит для тестирования пользовательского опыта, функциональности, безопасности, производительности. Не подходит, если необходимо проверить работу на разных версиях и видах операционных систем. Так же сложно применимо при сложной реализации логики и применении сложных алгоритмов.

## Автоматизированное тестирование программного обеспечения

Автоматизированное тестирование — это процесс проверки программного обеспечения, в котором тестовые сценарии выполняются с использованием специальных инструментов и скриптов, вместо ручного взаимодействия с приложением. Автоматизированное тестирование позволяет повторять тестовые сценарии, ускоряет процесс тестирования и улучшает его точность.

Преимущества автоматизированного тестирования мобильных приложений включают:

* Увеличение скорости и эффективности: Автоматизированное тестирование позволяет выполнять тестовые сценарии быстрее и более эффективно, по сравнению с ручным тестированием. *Даже ночью и во много потоков, что недоступно при ручном тестировании.*
* Повышение точности: Автоматизированное тестирование исключает человеческий фактор, что позволяет уменьшить вероятность ошибок и повысить точность результатов. *Человек же может допустить ошибку при проверке очередного тестового сценария.*
* Масштабируемость: Автоматизированное тестирование позволяет легко масштабировать тестовые сценарии для проверки на разных устройствах, операционных системах и разрешениях экрана. *Хороший плюс для борьбы с принципом о невозможности исчерпывающего тестирования*
* Повторяемость: Автоматизированные тесты могут быть повторно использованы для проверки приложения после каждого изменения или обновления. *Это является основной болью ручного тестировщика, ведь каждый раз нужно проверять одни и те же тестовые сценарии, чтобы убедиться, что они не были затронуты очередными изменениями*

Недостатки автоматизированного тестирования мобильных приложений включают:

* Сложность настройки: Настройка автоматизированного тестирования может быть сложной и требовать времени и ресурсов для создания и поддержки тестовых скриптов. *Соответственно, нужно учитывать временные трудозатраты на это.*
* Ограничения взаимодействия с пользователем: Автоматизированное тестирование может быть ограничено взаимодействием с пользователем, таким как проверка удобства использования и интерфейса приложения. *Поскольку машина на данный момент не может давать обратную связь насколько было удобно пользоваться готовым продуктом.*
* Необходимость обновления: Автоматизированные тесты требуют обновления при каждом изменении в приложении, чтобы они оставались актуальными и эффективными. *Например: поступила задача об изменении существующего модуля. Соответственно, для всех затронутых модулей могут потребоваться изменения написанных автоматизированных тестов.*

Цели и задачи автоматизированного тестирования мобильных приложений включают:

* Ускорение процесса тестирования: Автоматизированное тестирование позволяет выполнять тестовые сценарии быстрее, что ускоряет процесс разработки и выпуска приложения.
* Повышение качества: Автоматизированное тестирование помогает выявлять дефекты и ошибки в приложении, что позволяет повысить его качество.
* Повторяемость: Автоматизированные тесты могут быть повторно использованы для проверки приложения после каждого изменения или обновления.
* Тестирование различных сценариев: Автоматизированное тестирование позволяет проверить приложение на различные сценарии использования, включая краевые случаи и нестандартные ситуации.

Особенности автоматизированного тестирования мобильных приложений включают:

* Использование инструментов и скриптов: Автоматизированное тестирование требует использования специальных инструментов и скриптов для создания и выполнения тестовых сценариев.
* Тестирование на разных устройствах и операционных системах: Автоматизированное тестирование позволяет проверить приложение на разных устройствах, операционных системах и разрешениях экрана.
* Интеграция с CI/CD: Автоматизированное тестирование может быть интегрировано в процесс непрерывной интеграции и доставки (CI/CD), что позволяет автоматически выполнять тесты после каждого изменения в приложении.

Ограничения автоматизированного тестирования мобильных приложений включают:

* Сложность настройки: Настройка автоматизированного тестирования может быть сложной и требовать времени и ресурсов для создания и поддержки тестовых скриптов.
* Ограничения взаимодействия с пользователем: Автоматизированное тестирование может быть ограничено взаимодействием с пользователем, таким как проверка удобства использования и интерфейса приложения.
* Необходимость обновления: Автоматизированные тесты требуют обновления при каждом изменении в приложении, чтобы они оставались актуальными и эффективными.

Вывод:

Автоматизацию тестирования выгодно использовать для кроссплатформенного и регрессионного тестирования (в одном случае проверяется функциональность на всех возможных окружениях, в другом проверяется, что новые изменения не затронули уже стабильную функциональность). Автоматизацию невозможно использовать при тестировании пользовательского опыта.

При принятии решения об автоматизации тестирования необходимо учитывать трудозатраты на настройку и поддержание автоматизированных тестов в актуальном состоянии.

# 2 Анализ доступных средств автоматизации тестирования мобильных приложений

Эффективность автоматизации тестирования зависит не только от качества написанных тестов, но и от выбранного фреймворка для автоматизации тестирования. Выбор подходящего фреймворка зависит от специфики проекта, предпочтений команды разработчиков и требований к автоматизации тестирования.

Использование качественного и подходящего фреймворка позволит повысить эффективность и качество процесса автоматизации тестирования, обеспечивая надёжность и стабильность разрабатываемого программного обеспечения.  
 Для выбора фреймворка необходим анализ доступных средств автоматизации тестирования.

Выделяются ключевые рассматриваемые свойства:

* Поддерживаемые языки программирования;
* Возможность автоматизации мобильного тестирования;
* Сложность настройки;
* Устойчивость тестов;
* Стоимость;

Далее определяются значения свойств:

* Языки программирования – Python, Java либо Kotlin.
* Возможно написать автотесты под Android.
* Не требуется сложная настройка.
* Тесты устойчивы. Редко завершаются ошибкой без причин.
* Средство для автоматизации является бесплатным.

После того, как определены необходимые требования для инструмента автоматизации программного продукта, возможно приступить к анализу доступных средств для автоматизации.

## Драйверы

На сегодняшний день для автоматизации тестирования мобильных приложений существует несколько драйверов и большое множество надстроек. В целом все драйверы решают низкоуровневые проблемы: найти необходимый элемент, произвести с ним необходимые действия и выполнить необходимые проверки.

**Espresso:**

Espresso — это мощный фреймворк для автоматизации тестирования пользовательского интерфейса (UI) на платформе Android, который обеспечивает разработчикам инструменты и API для написания и выполнения автоматизированных тестов. Он поддерживает языки программирования Java и Kotlin, что делает его доступным для широкого круга специалистов.

Одним из ключевых преимуществ Espresso является его простота настройки, особенно для тех, кто уже имеет опыт в разработке приложений Android. Однако, как и с любым новым инструментом, для полного овладения всеми возможностями и функциями Espresso может потребоваться время и изучение документации.

Espresso обеспечивает высокую скорость и стабильность тестов по сравнению с другими инструментами. Это достигается за счет того, что все действия Espresso выполняются в том же процессе, где работает само приложение, что обеспечивает эффективную обработку команд. Действия и проверки преобразуются в сообщения и выполняются только в том случае, если приложение находится в состоянии ожидания пользовательского ввода.

Кроме того, Espresso является бесплатным и открытым инструментом с открытым исходным кодом, предоставляемым Google для разработчиков Android, что делает его доступным без каких-либо затрат. Это делает Espresso привлекательным выбором для автоматизации тестирования пользовательского интерфейса на платформе Android, обеспечивая удобство, эффективность и надежность в процессе разработки приложений.

**UiAutomator:**

UiAutomator — это инструмент автоматизации тестирования пользовательского интерфейса на платформе Android, который представляет собой мощное средство для разработчиков. Он входит в поставку Android SDK, начиная с версии API 16, и основан на механизме Binder IPC, что обеспечивает эффективное взаимодействие с элементами интерфейса.

Одним из ключевых преимуществ UiAutomator является возможность поиска элементов интерфейса, эмуляции различных действий (клики, тачи, свайпы, ввод текста) и проверки видимости элементов. Этот инструмент поддерживает языки программирования Java и Kotlin, что делает его доступным для широкого круга специалистов.

UiAutomator позволяет писать тесты по модели чёрного ящика, что упрощает процесс тестирования. Он работает в отдельном процессе и не требует доступа к исходному коду приложения, что делает его универсальным и готовым к взаимодействию с различными приложениями, включая системные.

Однако следует отметить, что механизм работы UiAutomator может быть нестабильным из-за ожидания консистентного состояния приложения и временного окна без системных событий. Это может привести к задержкам или полной остановке тестирования.

Несмотря на некоторые недостатки, UiAutomator остаётся важным инструментом для автоматизации тестирования пользовательского интерфейса на платформе Android, обеспечивая разработчикам удобный и эффективный способ проверки функциональности приложений.

**Selendroid и Robotium:**

Selendroid и Robotium — это два популярных инструмента для автоматизации тестирования мобильных приложений под управлением Android. Оба инструмента имеют свои особенности и преимущества, которые делают их привлекательными для разработчиков и тестировщиков.

Robotium, начиная с API level 8, предлагает широкий спектр возможностей для тестирования приложений на Android, включая работу с WebView начиная с API level 15. Этот инструмент позволяет писать тесты на Java, кроме того, Robotium обладает плагином Robotium Recorder для IntelliJ IDEA и Android Studio, что улучшает процесс создания баг-репортов.

С другой стороны, Selendroid, ограничиваясь версиями API от 10 до 19, предоставляет возможность использовать протокол WebDriver, что открывает доступ к использованию различных языков программирования для написания тестов. У Selendroid также есть удобная утилита Inspector, которая помогает визуально анализировать иерархию элементов и записывать record-and-playback-тесты.

Хотя оба инструмента имеют свои преимущества, стоит отметить, что они развиваются менее активно по сравнению с официальными драйверами от Google. Тем не менее, некоторые компании до сих пор предпочитают использовать Selendroid и Robotium из-за их удобства и функциональности. Важно помнить, что выбор инструмента для автоматизации тестирования зависит от конкретных потребностей проекта и предпочтений команды разработчиков.

**XCUITest:**

XCUITest — это инструмент для разработчиков, предоставляющий возможность писать и запускать автоматизированные UI тесты для приложений, созданных для операционных систем iOS, macOS, watchOS и tvOS. Он поддерживает два основных языка программирования: Swift и Objective-C, что делает его доступным для широкого круга специалистов.

Этот фреймворк обеспечивает разработчикам удобные инструменты для создания автоматических тестов, которые могут проверять функциональность и корректность кода в приложениях. Благодаря XCUITest можно проводить тестирование приложений без необходимости доступа к исходному коду, что упрощает процесс тестирования и повышает его эффективность.

С Xcode 9 драйвер XCUITest может тестировать несколько приложений одновременно, включая как сторонние, так и системные приложения. Хотя "из коробки" XCUITest позволяет запускать тесты только на симуляторах, существуют сторонние утилиты, позволяющие использовать этот инструмент и для работы с реальными устройствами, расширяя его возможности.

Одним из преимуществ XCUITest является его простота установки и использования. Фреймворк устанавливается вместе с Xcode, что делает его доступным и готовым к использованию без необходимости дополнительных настроек. Благодаря этому XCUITest становится удобным инструментом для автоматизации тестирования пользовательского интерфейса на платформах Apple.

XCUITest является бесплатным фреймворком с открытым исходным кодом, что делает его привлекательным выбором для разработчиков, стремящихся к эффективному и надежному тестированию приложений под управлением операционных систем Apple.

## 2.2 Надстройки

Надстройки помогают решать более высокоуровневые задачи. Например, в рамках одного и того же проекта реализовать разные проверки с помощью разных драйверов, реализовать кроссплатформенность и т. д.

**Appium**

Appium — это инструмент с открытым исходным кодом, который позволяет автоматизировать тестирование как десктопных, так и **мобильных приложений на платформах Android и iOS**. Его архитектура напоминает Selenium WebDriver, широко используемый стандарт в тестировании веб-приложений. Одним из ключевых преимуществ Appium является его кроссплатформенность, достигаемая благодаря использованию различных драйверов для разных платформ.

Appium предоставляет клиентские библиотеки с фиксированным API для написания тестов, что делает процесс автоматизации более удобным. Еще одним интересным аспектом является возможность использования разных драйверов в рамках одного проекта тестирования, что дает большую гибкость и возможность выбора подходящего инструмента для конкретной задачи. *Например: использовать UI Automator для проверки взаимодействия тестируемого приложения, а для тест-кейсов, не выходящих за рамки тестируемого приложения использовать более быстрый Espresso.*

Appium поддерживает множество популярных языков программирования, такие как **Java, PHP, Ruby, Objective-C, Python, JavaScript, C#,** что делает его доступным для широкого круга разработчиков. Однако, несмотря на все свои преимущества, у Appium есть и недостатки. Некоторые из них включают в себя **низкую скорость выполнения тестов, высокую хрупкость тестов и сложность настройки** из-за взаимодействия с различными драйверами.

Несмотря на эти ограничения, Appium остается популярным выбором для автоматизации тестирования мобильных приложений благодаря своей **бесплатности, открытому исходному коду** и широким возможностям.

**Kakao** Kakao является удобной надстройкой над Espresso, предназначенной для упрощения написания тестов и повышения их читаемости. Она поддерживает **язык программирования** **Kotlin**, что делает ее привлекательным выбором для разработчиков, использующих этот язык.

В Kakao реализованы два основных паттерна: KView и Screen. KView представляет собой Kakao-представление элемента интерфейса, с которым взаимодействует тест. Этот паттерн упрощает работу с элементами интерфейса, такими как текстовые поля (KEditText), кнопки (KButton) и другие. Пользователь также может создавать собственные реализации KView.

Screen, в свою очередь, представляет собой реализацию паттерна PageObject из мира веб-разработки. Это базовый класс, на основе которого создаются stateless-хранилища всех KView на соответствующих экранах приложения. Рекомендуется создавать отдельный Screen для каждой активити, фрагмента или другого элемента приложения, что способствует более четкой организации тестов.

**Сложность настройки Kakao не превышает сложность настройки Espresso, а использовать его гораздо проще.**

По сравнению с Appium, Kakao представляет собой более легковесную обертку, не добавляющую значительного оверхеда к вашим тестам. Это позволяет командам разработчиков **э**ффективно выполнять тесты в главном потоке приложения. **Благодаря наследованию особенностей Espresso**, Kakao **сохраняет стабильность тестов и обеспечивает хорошую производительность**.

Кроме того, Kakao является **проектом с открытым исходным кодом и доступен бесплатно**, что делает его доступным для широкого круга разработчиков. В целом, Kakao представляет собой мощный инструмент для автоматизации тестирования **мобильных приложений на Android**, облегчая процесс написания и поддержки тестовых сценариев.

**Kasspresso**  
 Kaspresso - это еще одна удобная надстройка над Espresso, которая предоставляет дополнительные возможности и функциональность для тестирования мобильных приложений на платформе Android. Он объединяет в себе функционал Espresso и UiAutomator, что делает его универсальным инструментом для написания тестов как в контексте приложения, так и за его пределами.

Одним из ключевых преимуществ Kaspresso является возможность конфигурации и расширения функционала через использование интерсепторов различных типов. Это позволяет внедрять собственный код для обработки событий, таких как действия с элементами интерфейса или проверки состояний во время выполнения тестов. Кроме того, все параметры могут быть легко настроены перед запуском теста, что обеспечивает гибкость и контроль над функциональностью.

Kaspresso также отличается возможностью автоматического повторения действий или проверок в случае их неудачного выполнения, закрытия системных диалогов и автоматической прокрутки экрана до нужного элемента внутри ScrollView. Эти функции способствуют повышению стабильности тестов и обеспечивают более надежное тестирование приложений.

Несмотря на свою мощную функциональность, сложность настройки Kaspresso не превышает сложности настройки Espresso и UiAutomator, что делает его отличным выбором для команд разработчиков, стремящихся к улучшению процесса тестирования. Как и Kakao, Kaspresso является проектом с открытым исходным кодом, доступным бесплатно для использования.

* Поддерживаемые языки программирования;
* Возможность автоматизации мобильного тестирования;
* Сложность настройки;
* Устойчивость тестов;
* Стоимость;

Далее определяются значения свойств:

* Языки программирования – Python, Java либо Kotlin.
* Возможно написать автотесты под Android.
* Не требуется сложная настройка.
* Тесты устойчивы. Редко завершаются ошибкой без причин.
* Средство для автоматизации является бесплатным.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Фреймворк | Языки программирования | Автоматизация мобильных приложений | Сложность настройки | Устойчивость тестов/  Скорость | Стоимость |
| Espresso | Java | Android | Не требует настройки | Нет встроенной защиты от случайных падений тестов /Высокая | Бесплатнен |
| UI Automator | Java/Kotlin | Android | Не требует настройки | Нет встроенной защиты от случайных падений тестов/Низкая | Бесплатнен |
| Appium | Почти все современные языки | Android/iOS | Высокая | Нет встроенной защиты от случайных падений тестов/Низкая | Бесплатнен |
| Selendroid/Robotium | Java | Android | - | Нет встроенной защиты от случайных падений тестов/Средняя | Бесплатнен |
| XCUITest | Swift/Objective-C | iOS | Не требует настройки | - | Бесплатнен |
| Kakao | Kotlin | Android | Низкая | Нет встроенной защиты от случайных падений тестов/Высокая | Бесплатнен |
| Kasspresso | Kotlin | Android | Низкая | Есть встроенная защита от случайных падений/Высокая | Бесплатнен |

Таким образом на данный момент оптимальным решением при заданных условиях является фреймворк Kasspresso. Он выделятся простым синтаксисом, высокой устойчивостью UI-тестов, использованию более современного Kotlin, вместо Java, так же не уступает по скорости тестов Espresso, является бесплатным и легко встраивается в проект.

# Создание мобильного приложения

# Библиографический список

1. ISO/IEC TR 19759:2015. Разработка программного обеспечения. Руководство к Своду Знаний по программной инженерии (SWEBOK)
2. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology
3. Скрябин А. М., Кардаш Д. И. Жизненный цикл композиционно-адаптируемого программного обеспечения // Аспирант и соискатель. 2008. № 2. С. 171-174.

1. ISO/IEC TR 19759:2015. Разработка программного обеспечения. Руководство к Своду Знаний по программной инженерии (SWEBOK) [↑](#footnote-ref-1)
2. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology [↑](#footnote-ref-2)
3. Скрябин А. М., Кардаш Д. И. Жизненный цикл композиционно-адаптируемого программного обеспечения // Аспирант и соискатель. 2008. № 2. С. 171-174. [↑](#footnote-ref-3)