A E R O

Appium vs Нативные инструменты разработки мобильных автотестов:

влияние выбора инструмента на процесс разработки и тестирования

О нас

Агентство Aero

16+ лет успешных проектов и сбора лучших практик рынка РФ и за рубежом

N°1 в России по разработке больших интернет-магазинов и маркетплейсов

Огромный опыт в разных индустриях, B2C- и B2B-сегментах

50+ млрд
выручка целевого клиента Aero

Работаем собственными ресурсами.

Практика общих проектных команд

Совокупная выручка за 2022 год

500+ млн
рублей

В штатной команде

100+ чел.

Сильная продуктовая и IT-экспертиза, подход для проектов

от 50+ млн
рублей

Быстрый запуск MVP и agile-подход в работе над проектом

О спикерах

Роман

14 лет **в управлении** IT проектами.

С 2017 управление проектами по автоматизации тестирования и лидирование стрима тестирования. С 2020 в партнерстве с **Aero** развиваю комьюнити экспертов в формате платформы развития талантов с ориентацией на интересы Senior инженеров (DevOps, QAA, SA, Dev…). **Верю в людей.**

Александр

Senior QA Automation Engineer. В IT работаю с 2012 года, тестированием начал заниматься в 2015 году и продолжаю до сих пор. Нравится работать над большими и сложными продуктами, которыми пользуются большое количество людей и видеть результат своей работы.

Илья

Lead QA Engineer, в тестировании 9 лет.

Считаю, что стек инструментов не так важен, как понимание решения проблемы. В Aero мы ценим людей, которые не боятся использовать разный технологический стек и брать на себя ответственность за результат.

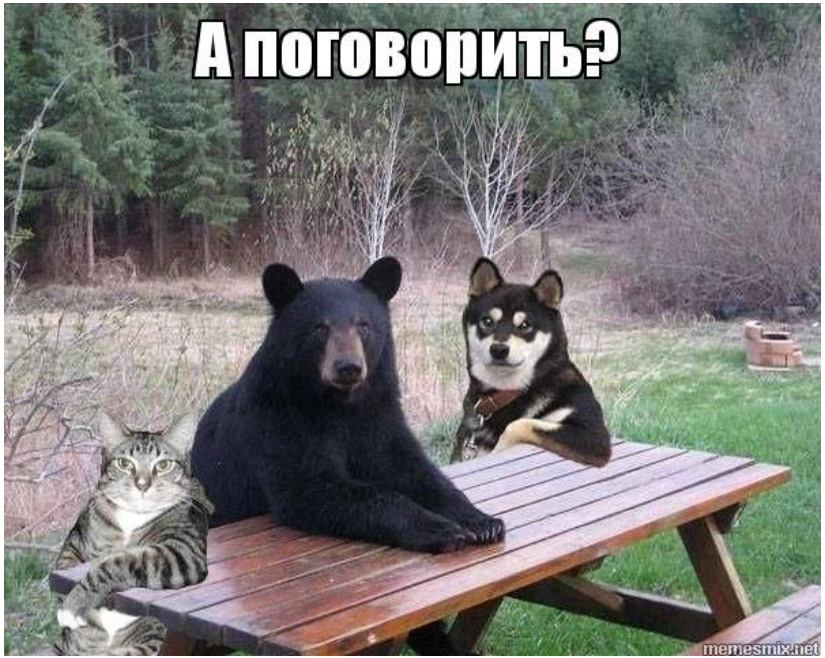
План выступления

1. Почему мы решили об этом поговорить
2. Проблемы тестирования мобильных приложений
3. Чего хотят заказчики
4. Анализ инструментов автоматизации
5. Какие задачи приходилось решать
6. Влияние инструмента на процессы внутри команды
7. Анализ рынка Mobile QA Automation
8. Выводы

1. Почему мы решили об этом поговорить

Почему мы решили об этом поговорить

1. Мобильные приложения в тренде
2. Осознание необходимости выстраивания процесса тестирования у заказчика
3. Заказчики хотят тестировать мобильные приложения не только вручную
4. Заказчики задают вопросы
5. Заказчики не знают чего хотят



2. Проблемы тестирования мобильных приложений

Проблемы тестирования мобильных приложений

1. Различные платформы и девайсы (необходимость запускать одни и те же тесты для разных платформ)
2. Отсутствие тестовой инфраструктуры для кроссплатформенного тестирования ограничение только теми устройствами, которые есть у тестировщиков лично
3. Различные типы приложений: **web / нативные / гибридные**
4. Различные тестовые интеграции

3. Чего хотят заказчики

Чего хотят заказчики

1. Поставлять качественные приложения
2. Сократить длительные циклы ручного тестирования
3. Сделать тестирование менее зависимым от человеческого фактора
4. Проверять работоспособность приложений на большем количестве устройств
5. Сократить TTM
6. Отсутствие подходов к централизованному хранению тест-кейсов и управлению циклами тестирования
7. Низкая зрелость процессов разработки и тестирования

4. Анализ инструментов автоматизации

Основные инструменты на рынке

1. Appium
2. XCUITest
3. Espresso
4. Все остальное

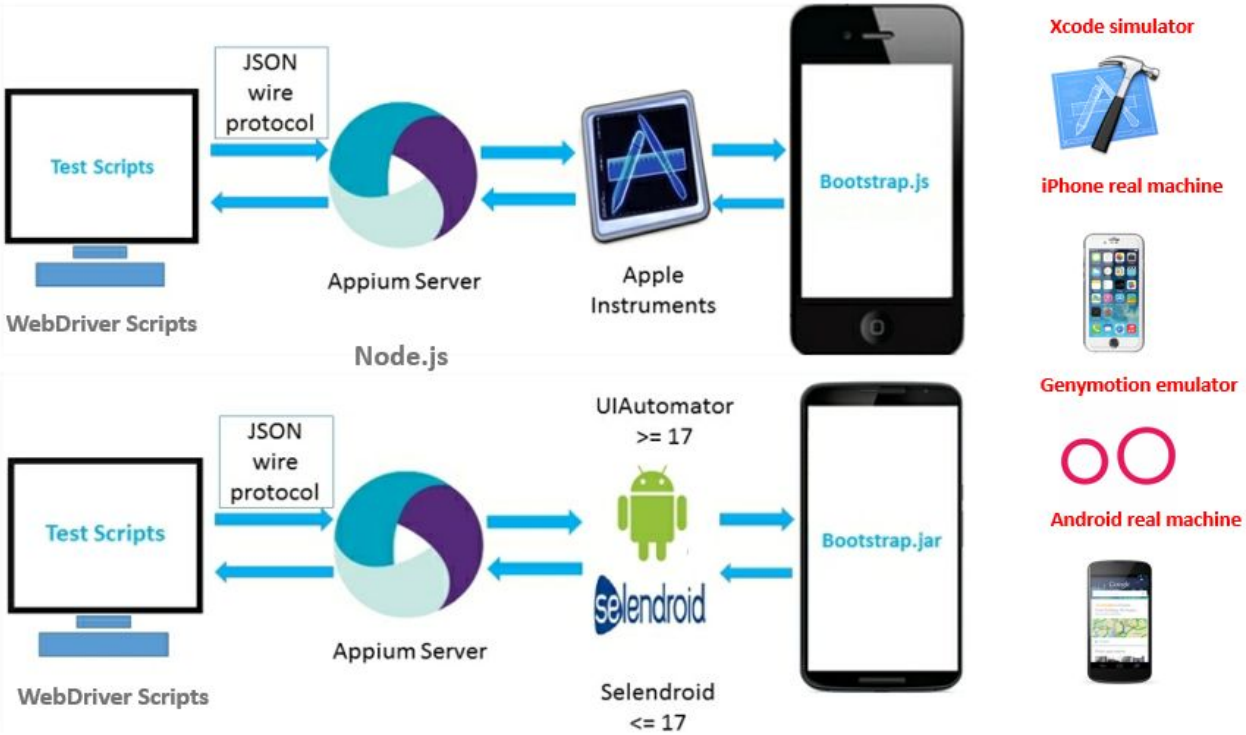
Сравнение инструментов для автоматизации тестирования

	Appium	Espresso	XCUITest
Цель использования	Используется для автоматизации тестирования мобильных приложений на Android и iOS.	Используется только для автоматизации тестов пользовательского интерфейса на Android.	Используется для автоматизации тестов пользовательского интерфейса для приложений iOS.
Покрытие	Поддерживает тестирование как мобильных, так и десктоп приложений.	Поддерживает тестирование мобильных приложений на Android	Поддерживает тестирование мобильных приложений на IOS
Метод тестирования	Black box	White box	White box
Языки программирования	Поддерживает несколько языков, таких как Java, Python, Ruby, PHP, C# и некоторые другие.	Поддерживает только Java и Kotin.	Поддерживает только Swift и Objective C.
Установка	Сложно	Легко	Легко

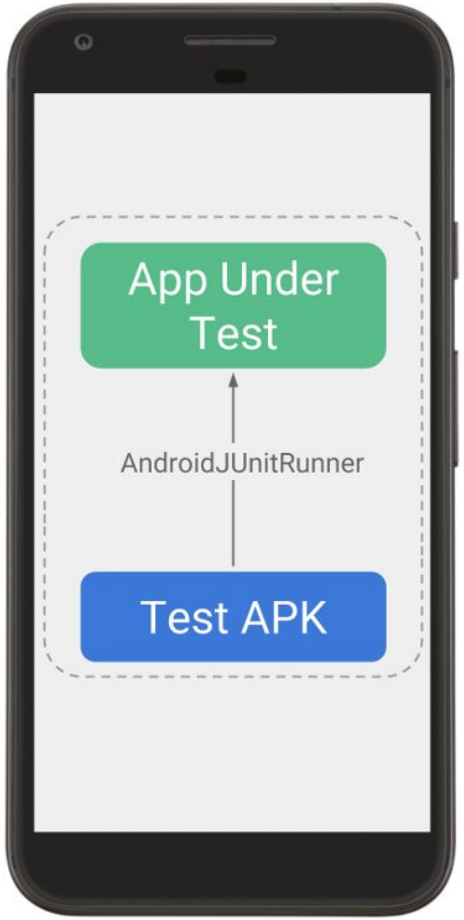
Appium

Appium on iOS

Appium on Android



Нативные инструменты



Пример теста Espresso

```
@get:Rule
val activityRule = ActivityScenarioRule(LoginActivity::class.java)

@Test
fun baseChecks() {
    onView(withId(R.id.emailEditTextField))
        .perform(typeText( stringToBeTyped: "test@email.com"))
        .check(matches(isDisplayed()));

    onView(withId(R.id.passwordTextField))
        .perform(typeText( stringToBeTyped: "myStrongPassword"))
        .check(matches(isDisplayed()));

    onView(withId(R.id.loginButton))
        .perform(click())
        .check(matches(isDisplayed()));
}
```

Пример теста XCUITest

```
let app = XCUIApplication()
app.launch()

let emailField = app.textFields["Email"]
emailField.tap()

let clearEmailFieldButton = app.buttons["Закрыть"]
clearEmailFieldButton.tap()
emailField.typeText("alexsimplestar@gmail.com")

let passwordField = app.secureTextFields["Password"]
passwordField.tap()
passwordField.typeText("1234Qwer!")

let loginButton = app.buttons["Log in"]
loginButton.tap()
```

Преимущества нативных инструментов

1. Работают в контексте приложения
2. Стабильность и скорость работы
3. Возможность перехода к конкретному экрану
4. Возможность тестирования нескольких приложений сразу
5. Нативная работа с **gestures** - swipe, scroll, tap, pinch (zoom) и другие
6. Возможность передавать в приложение параметры при запуске теста
7. Поддержка продвинутых запросов к элементам
8. Возможность визуальных проверок

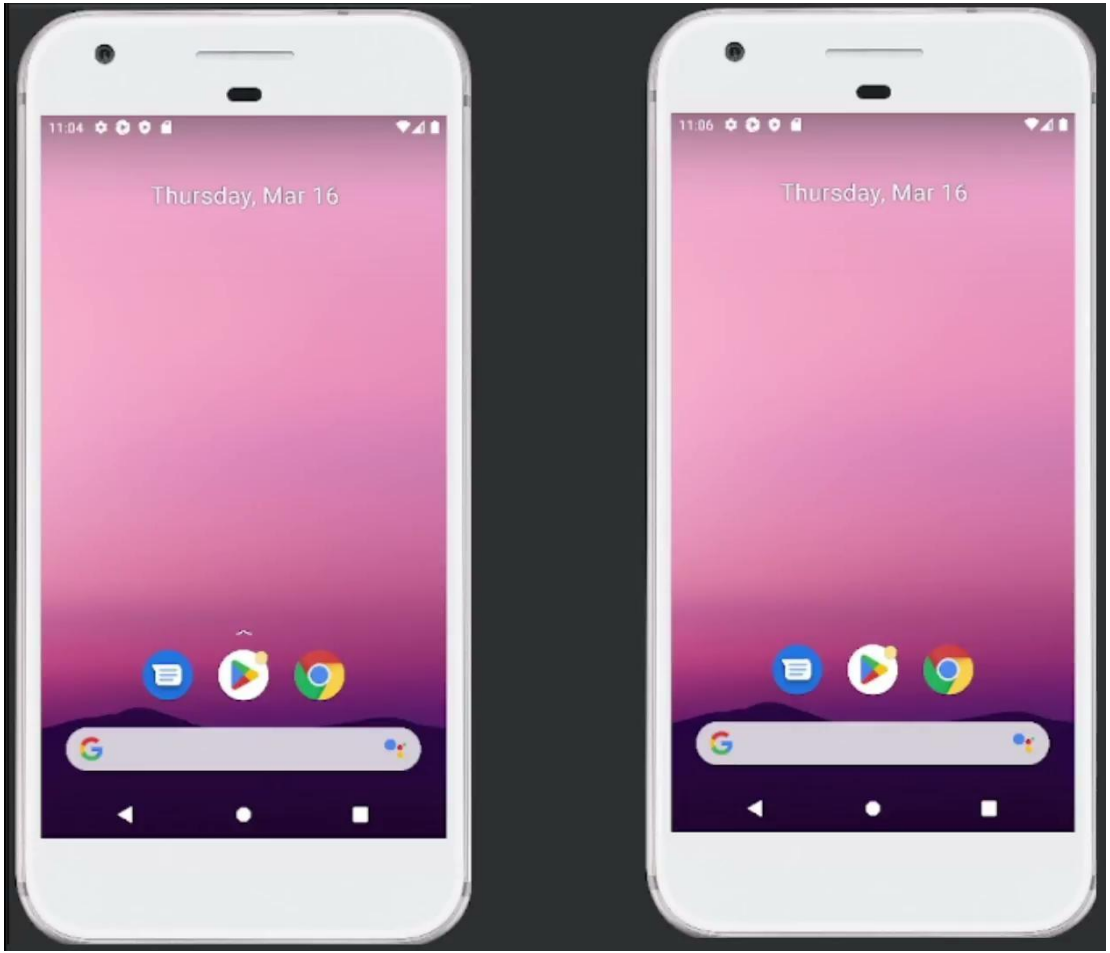
Поддержка продвинутых запросов к элементам

```
app.buttons.element

// the button titled "Help"
app.buttons["Help"]

// all buttons inside a specific scroll view (direct subviews only)
app.scrollViews["Main"].children(matching: .button)

// all buttons anywhere inside a specific scroll view (subviews, sub-subviews, sub-sub-subviews, etc)
app.scrollViews["Main"].descendants(matching: .button)

// find the first and fourth buttons
app.buttons.element(boundBy: 0)
app.buttons.element(boundBy: 3)

// find the first button
app.buttons.firstMatch
```

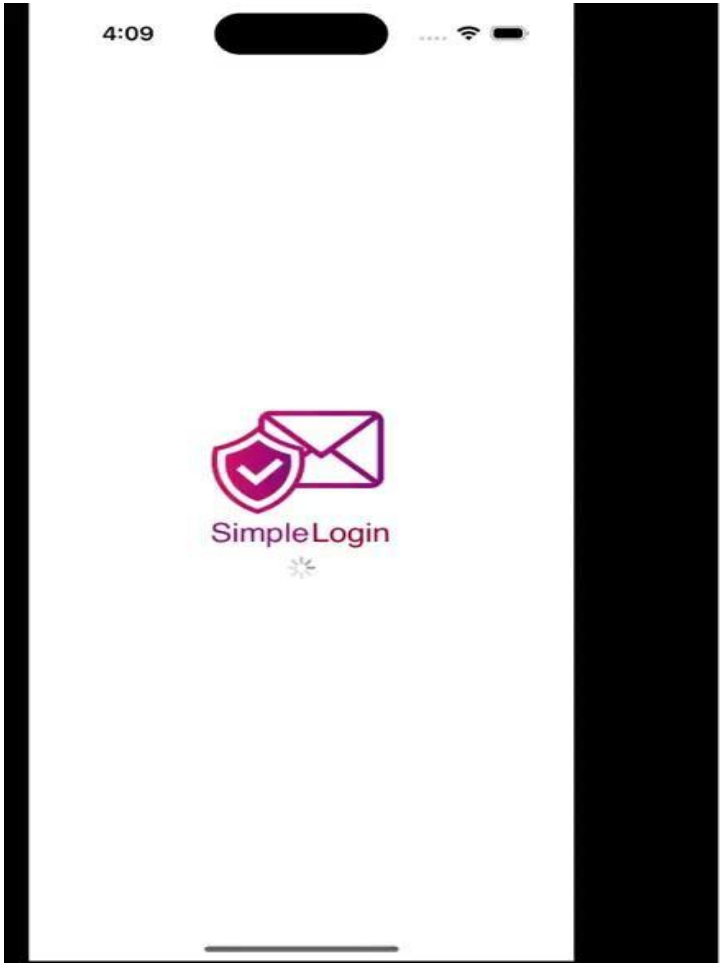
Визуальные проверки в Espresso

```
onView(withId(R.id.loginButton)).check(isCompletelyBelow(withId(R.id.emailEditTextField)))

onView(withId(R.id.emailEditTextField)).check(isCompletelyAbove(withId(R.id.passwordEditTextField)))

onView(withId(R.id.emailEditTextField)).check(isCompletelyLeftOf(withId(R.id.passwordEditTextField)))

onView(withId(R.id.emailEditTextField)).check(isCompletelyRightOf(withId(R.id.passwordEditTextField)))

onView(withId(R.id.emailEditTextField)).check(isRightAlignedWith(withId(R.id.passwordEditTextField)))

onView(withId(R.id.emailEditTextField)).check(isTopAlignedWith(withId(R.id.passwordEditTextField)))

onView(withId(R.id.emailEditTextField)).check(noEllipsizedText())

onView(withId(R.id.passwordEditTextField)).check(noMultilineButtons())

onView(withId(R.id.loginButton)).check(noOverlaps())
```

Appium vs Espresso



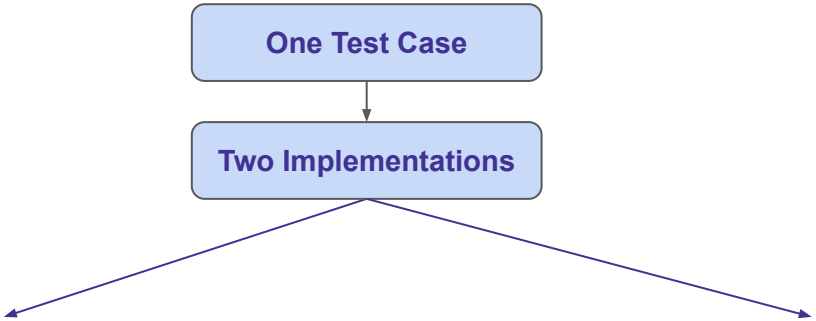
Appium vs XCUITest



5. Какие задачи приходилось решать

Использование одного теста для запуска на разных платформах

Фреймворк на основе Java для автоматизации тестирования.

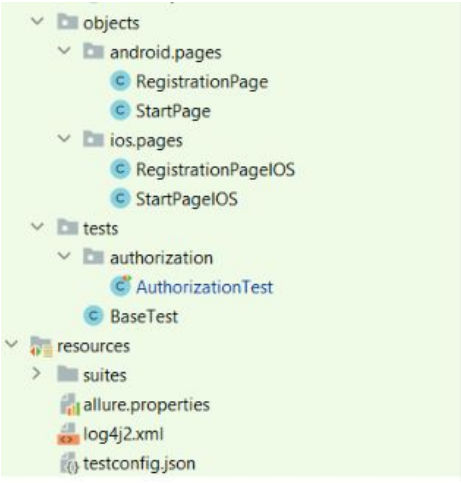



```java
@Step("Initial screen should be opened")
public StartPage shouldBeOpen() {
    welcomeButtons.shouldBe(visible);
    return this;
}
```

```java
@Step("Initial screen should be opened")
public StartPageIOS shouldBeOpen() {
    welcomeButtons.shouldBe(visible);
    moreOptions.shouldBe(visible);
    return this;
}
```

Использование одного теста для запуска на разных платформах

```
Class<T> instanceClass = objectClass;
String instanceClassName = objectClass.getName();
try {
    if (TestConfiguration.isIOS()) {
        instanceClassName = instanceClassName.replaceFirst( regex: ".android.", String.format(".%s.", "ios"))
            + "IOS";
    }
    instanceClass = (Class<T>) Class.forName(instanceClassName);
```




```
19  @Test(description = "Check user can proceed to registration screen")
20  @AllureId("2")
21  @Tag("FM03")
22  public void openRegistration() {
23      StartPage cartPage = instance(StartPage.class);
24      RegistrationPage registrationPage = instance(RegistrationPage.class);
25      cartPage
26              .shouldBeOpen()
27              .clickMoreOptions()
28              .clickEmailOption();
29      registrationPage
30              .shouldBeOpen();
31  }
32
33
```

Визуальное тестирование

1. Единый инструмент визуальной регрессии
2. Поддержка тестирования web
3. Добавить поддержку мобильных устройств
4. Функционал:
    a. Полностраничные скриншоты
    b. Отдельные элементы
    c. Разные состояния элемента

Hermione

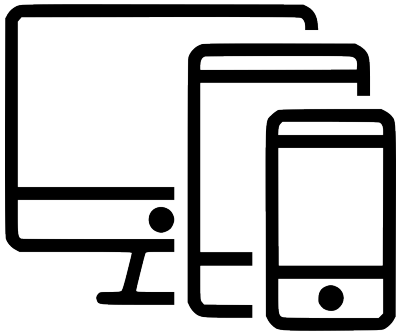
https://github.com/gemini-testing/hermione

1. Создана Yandex
2. На основе понятных инструментов
   a. Webdriver IO
   b. Mocha
3. Есть
   a. Полноэкранные скриншоты
   b. Выкалывание элемента
   c. Поддержка состояний
4. Нет
   a. Поддержки мобильных устройств
   b. Allure TestOps

Разработали плагины

1. **Mobile-Screenshot Plug-in**

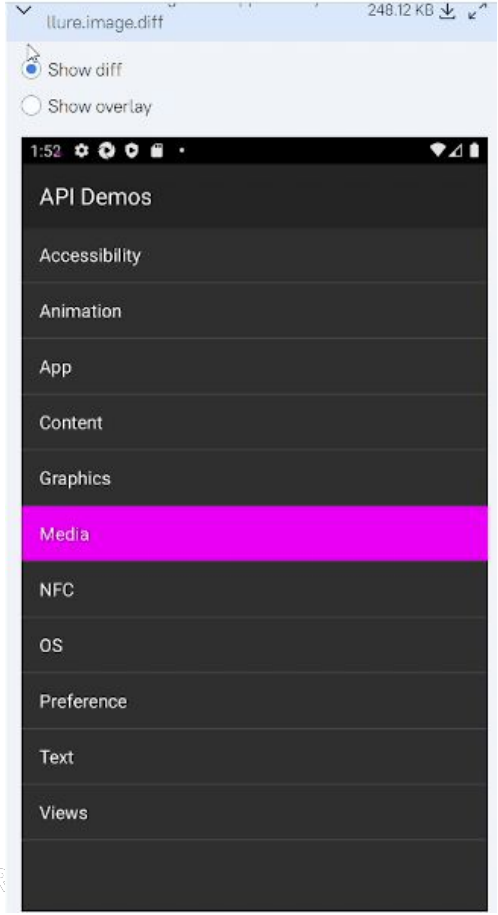
   Через Appium позволяет делать скриншоты экранов и отдельных элементов на мобильных устройствах



2. **Allure-Testops Plug-in**

   Интеграция с allure для выгрузки отчетов

Примеры


llure.image.diff
248.12 KB
Show diff
Show overlay
1:52
API Demos
Accessibility
Animation
App
Content
Graphics
Media
NFC
OS
Preference
Text
Views


Content logo ImageDiff application/vnd.
allure.image.diff
17.08 KB
Show diff
Show overlay
Content

Проблема Flutter приложений

1. Приложение написано на Flutter
   a. нельзя использовать Appium Inspector для просмотра дерева элементов, если нет доступа к исходному коду
2. Flutter Driver
   a. ограниченный набор **finders**
   b. ограниченный набор команд в принципе
3. Appium Flutter Driver
   a. существует реализация только для JavaScript
   b. только стандартный набор **finders** & **commands** Flutter Driver
   c. нет возможности писать расширения

Кастомные Finders

Для реализации кастомных **finder** необходимо реализовать два абстрактных класса

```
class PageKeyFinder extends SerializableFinder {
  const PageKeyFinder(this.pageKey);

  final String pageKey;

  @override
  String get finderType => 'ByPageKey';

  @override
  Map<String, String> serialize() => super.serial
    ..addAll(
      <String, String>{
        'pageKey': pageKey,
      },
    );
}
```

```
class PageKeyFinderExtension extends FinderExtension {
  @override
  String get finderType => 'ByPageKey';

  @override
  SerializableFinder deserialize(
      Map<String, String> params, DeserializeFinderFactory finderFactory) {
    return PageKeyFinder(params['pageKey']!);
  }

  @override
  Finder createFinder(
      SerializableFinder finder, CreateFinderFactory finderFactory) {
    final PageKeyFinder pageKeyFinder = finder as PageKeyFinder;

    return find.byElementPredicate(
      (Element element) {
        final Widget widget = element.widget;
        if (widget.key is PageKey) {
          return (widget.key as PageKey).value == pageKeyFinder.pageKey;
        }

        return false;
      },
    );
  }
}
```

Кастомные Commands

Нельзя использовать стандартный механизм расширения **Flutter Driver** с Appium Flutter Driver (нет возможности отправки запроса из Appium на выполнение **custom command**)

Поэтому мы придумали:

1. Реализовывать работу с командами с помощью переопределенной команды **RequestData**
2. Для реализации **custom command** реализовать два класса: **CommandWithTarget** и **CommandExtension**

Кастомные Commands

```
class RequestDataCommand extends CommandWithTarget {
  RequestDataCommand(SerializableFinder finder, this.data, {Duration? timeout})
      : super(finder, timeout: timeout);

  RequestDataCommand.deserialize(…

  final Map<String, String> data;
  late SerializableFinder scrollable;

  static const String messageFieldName = 'message';
  static const String kindName = 'request_data';

  @override
  Map<String, String> serialize() {…

  @override
  String get kind => kindName;
}

class RequestDataCommandResult extends Result {…
```

```
class RequestCommandExtension extends CommandExtension {
  @override
  String get commandKind => RequestDataCommand.kindName;

  @override
  Future<Result> call(
    covariant RequestDataCommand command,
    WidgetController prober,
    CreateFinderFactory finderFactory,
    CommandHandlerFactory handlerFactory,
  ) async {
    final String? dataCommand = command.data['command'];

    switch (dataCommand) {
      case 'mycommad1':
        return handlerFactory.handleCommand(…
      case 'mycommad2':
        return handlerFactory.handleCommand(…
    }

    throw DriverError('Unsupported command: $dataCommand');
  }

  @override
  Command deserialize(…
    return RequestDataCommand.deserialize(data, finderFactory);
  }
}
```

Подключение кастомных Finders & Commands

1. Перед **runApp** необходимо вызвать метод **enableFlutterDriverExtension()**
2. В нем подключить **кастомные Finders & Commands**

```
void enableFlutterDriver() {
  enableFlutterDriverExtension(
    finders: <FinderExtension>[
      PageKeyFinderExtension()
    ], // <FinderExtension>[]
    commands: <CommandExtension>[
      RequestCommandExtension(),
      ScrollUntilVisibleCommandExtension(),
      TapTextSpanCommandExtension(),
      WaitForStateCommandExtension(),
      GetWidgetStateCommandExtension()
    ], // <CommandExtension>[]
  );
}
```

6. Влияние инструмента на процессы внутри команды

Доступность на рынке и вовлеченность в процессы

1. Доступность на рынке и стоимость специалистов
2. Вовлеченность команды в процессы разработки и коммуникации
   a. **Appium:** низкая или средняя, возможно работать изолированно
   b. **Нативные инструменты**: высокая (ревью, менторинг, онбординг)

Риски с точки зрения менеджмента

1. **Appium**
   a. медленное покрытие на старте
   b. медленное прохождение AT
   c. тестовый код и код разработки не вместе
   d. тесты не поддерживаются, не используются, нет доверия
   e. тесты **нестабильны**
2. **Нативные инструменты**
   a. большая трата ресурсов команды разработки
   b. взаимозаменяемость (особенно если учесть отдельно что нужен и ios и android)
   c. повышенные требования к кандидатам и высокая стоимость развития компетенций внутри
   d. менее широкие возможности применения, если у QA недостаточно опыта

7. Анализ рынка Mobile QA Automation

Средние зарплаты

Как отличаются зарплаты QA Automation с Appium от специалистов с нативными инструментами?

Средние зарплаты

Как отличаются зарплаты QA Automation с Appium от специалистов с нативными инструментами?

1. **Middle**

Средние зарплаты

Как отличаются зарплаты QA Automation с Appium от специалистов с нативными инструментами?

1. **Middle** < на 30 тысяч рублей

Средние зарплаты

Как отличаются зарплаты QA Automation с Appium от специалистов с нативными инструментами?

1. **Middle** < на 30 тысяч рублей
2. **Senior**

Средние зарплаты

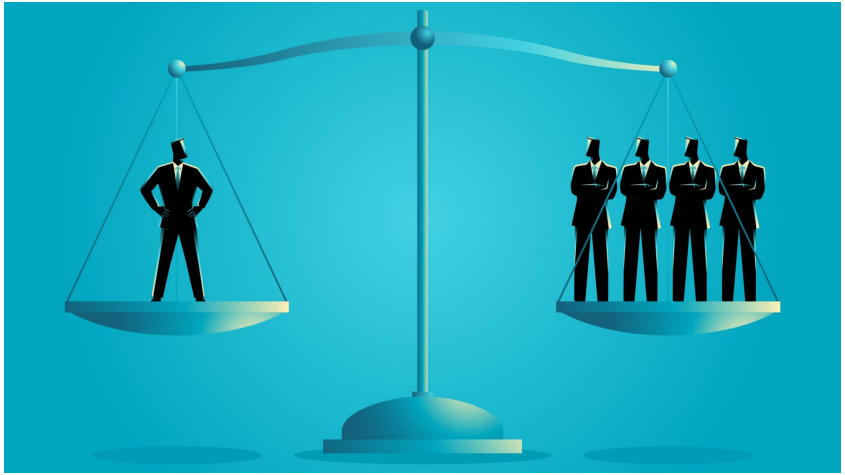
Как отличаются зарплаты QA Automation с Appium от специалистов с нативными инструментами?

1. **Middle** < на 30 тысяч рублей
2. **Senior** < на 60 тысяч рублей

* Выборка на 300 случайных резюме специалистов с компетенцией автоматизации мобильного тестирования

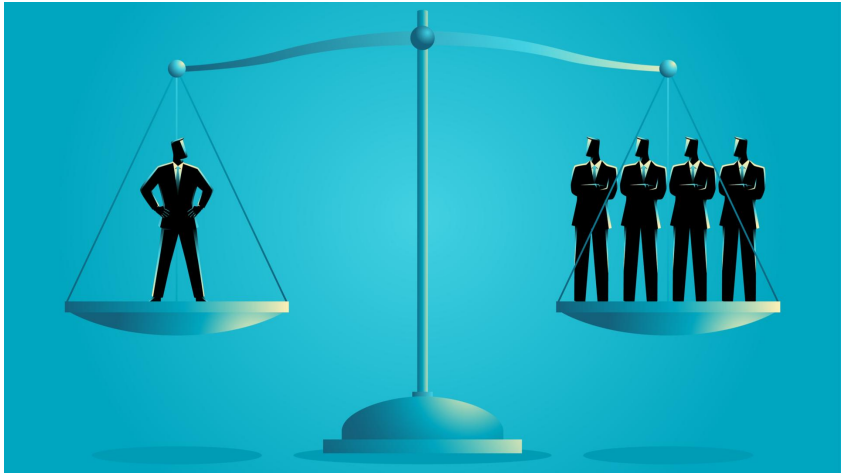
Рынок специалистов

1. Специалисты с **Appium** встречаются ….



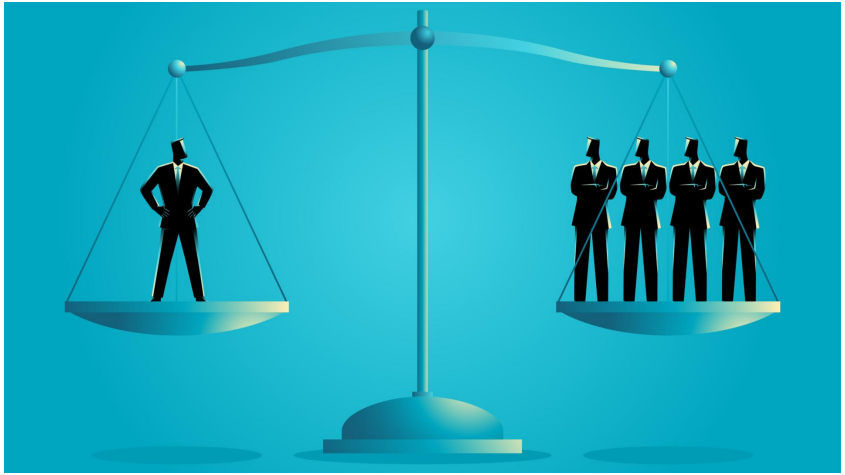
Рынок специалистов

1. Специалисты с **Appium** встречаются в **5 раз** чаще, чем с опытом нативных инструментов



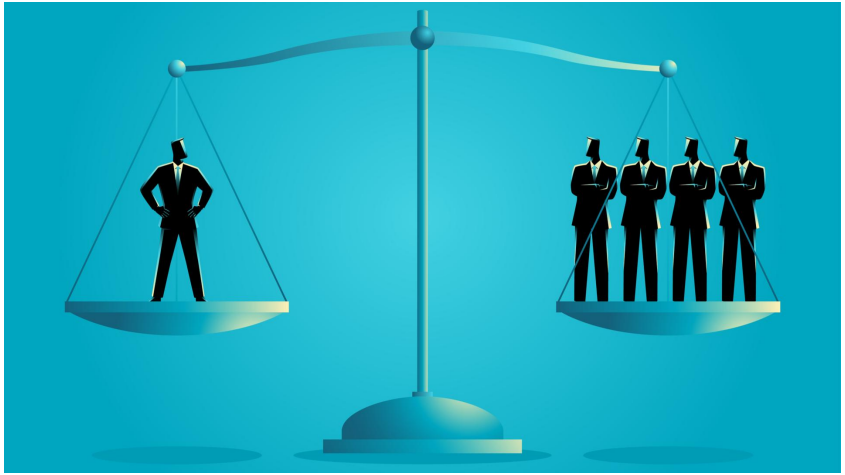
Рынок специалистов

1. Специалисты с **Appium** встречаются в **5 раз** чаще, чем с опытом нативных инструментов
2. **???** специалистов с **Appium** также имеют опыт с нативными инструментами



Рынок специалистов

1. Специалисты с **Appium** встречаются в **5 раз** чаще, чем с опытом нативных инструментов
2. **20%** специалистов с **Appium** также имеют опыт с нативными инструментами



* Выборка на 300 случайных резюме специалистов с компетенцией автоматизации мобильного тестирования

Опыт

Опыт использования нативных инструментов

1. **ЧАЩЕ** встречается у специалистов в тестировании с опытом **от 6 лет**

2. **РЕЖЕ** - с опытом **1 - 3 года**

* Выборка на 300 случайных резюме специалистов с компетенцией автоматизации мобильного тестирования

8. Выводы

Выводы

1. Выбор инструмента для автоматизированного тестирования **мобильных приложений** зависит от: **состава** команды, **зрелости** процессов, **потребностей**
2. Порог вхождения ниже на Appium
3. Для тестирования одной платформы лучше подходят нативные инструменты
4. Использование нативных инструментов тесно связывает разработку и тестирование, что может по-разному влиять на процесс в целом
5. Скорость работы тестов на нативных инструментах выше
6. Специалисты с Appium шире представлены на рынке

В этом докладе мы обсудили

1. Почему мы решили об этом поговорить
2. Проблемы тестирования мобильных приложений
3. Чего хотят заказчики
4. Анализ инструментов для автоматизации
5. Какие задачи приходилось решать
6. Влияние инструмента на процессы внутри команды
7. Анализ рынка Mobile QA Automation

Ссылки

Пример **Kotlin/Kaspresso**

Пример **XCUITest/Swift**

Примеры с Flutter и написанием одного кейса на несколько платформ можно запросить дополнительно по email: **smotrov@aeroidea.ru**

Спасибо за внимание!