Software Testing

Vorlesung – Mobile Testing

WS2020 - 183.290

Christoph Hafner & Michael Wapp

Lecture starts at 17:05

peso PERFECTING SOFTWARE

Outline

- Mobile Basics
- Challenges & Risks of Mobile Development
- Test Types
- Platform Tools & Environments
- Test Software & Frameworks
- Mobile Test Automation

Mobile Basics

Mobile Device Types

- Basic phones
- Feature phones
- Smartphones
- Tablets
- Companion devices
  - Wearable devices (e.g. smartwatches), some IoT devices

Mobile Application Types

- Native applications
  - Use platform specific software development kits (SDKs), dev tools and features
  - Recommended approach by manufacturers (and often required)
- Browser-based applications
  - PWAs (Progressive Web Apps)
  - Mobile optimized homepages [m(dot)-sites]
  - Responsive homepages
- Hybrid applications
  - Javascript-based (React Native, Ionic, ...)
  - Flutter
  - Xamarin (.NET-Framework)

Challenges & Risks of Mobile Development

Challenges & Risks of Mobile Development

- Hardware differences
- Screen sizes

Challenges & Risks of Mobile Development

- Hardware differences
  - Screen sizes
  - Sensors
  - Wireless communications (Cellular, WiFi, Bluetooth, etc.)
  - Input/Output options
  - (Foldable)
- Software differences
- Interruptions
  - Network interruptions
  - Lifecycle interruptions
- New devices and features introduced regularly
- Too many other influences on the application

Test Types

“Common” Test Types

- Stress testing
  - Limited resources on device
- Security testing
  - Code injections, decompilation, unencrypted/sensitive data, stored keys
- Performance testing
  - 3 seconds rule
- Database testing
  - Validation of data stores local/remote, third-party read/write
- Globalization and Localization testing
  - Left2Right/Right2Left, date-formats, string dimensions
- Accessibility & Usability testing
  - Guidelines by Apple/Google

Mobile Test Types

- Installability testing
  - Installation/Updates/De-Installation
- Testing for various connectivity methods
- Testing for device hardware compatibility
- Testing for app interactions with device software

Testing for compatibility with device hardware

- Use a selected subset of devices for testing the app

- Testing for …
  - … different device features (like USB, cameras, microphone, etc.)
  - … different displays
  - … different sensors
  - … different I/O - types
  - … typical interrupts
  - … access permissions
  - … power consumption

Testing for app interactions with device software

- Testing for ...
  - ... notifications
  - ... user preferences
  - ... co-existence with other apps
  - ... testing for different types of apps:
    - Native apps: device compatibility
    - Hybrid apps: Interaction with device features, look-and-feel
    - Web apps: Cross-browser compatibility, look-and-feel, JavaScript engine compatibility

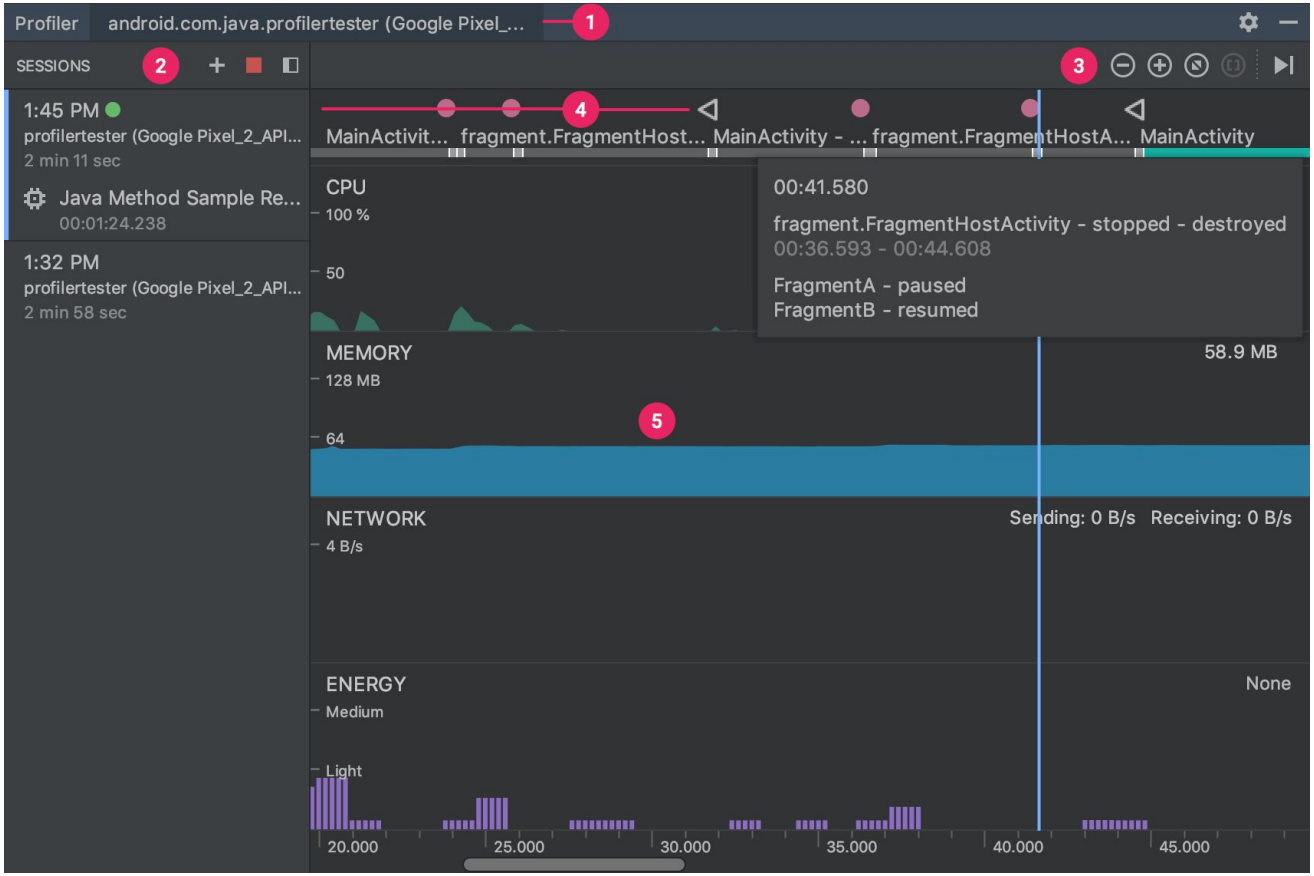
Platform Tools & Environments

Platform Tools & Environments

- Tools provided in Software Development Kits (SDK)

- Android - Android Studio (IDE)
  - Android SDK (currently Version 30)
  - Tools:
    - Android Debug Bridge (adb)
    - Systrace tool (systrace)
    - Fastboot (fastboot)
  - Android Profiler - App Performance Measurement Tool
  - Android Layout Inspector

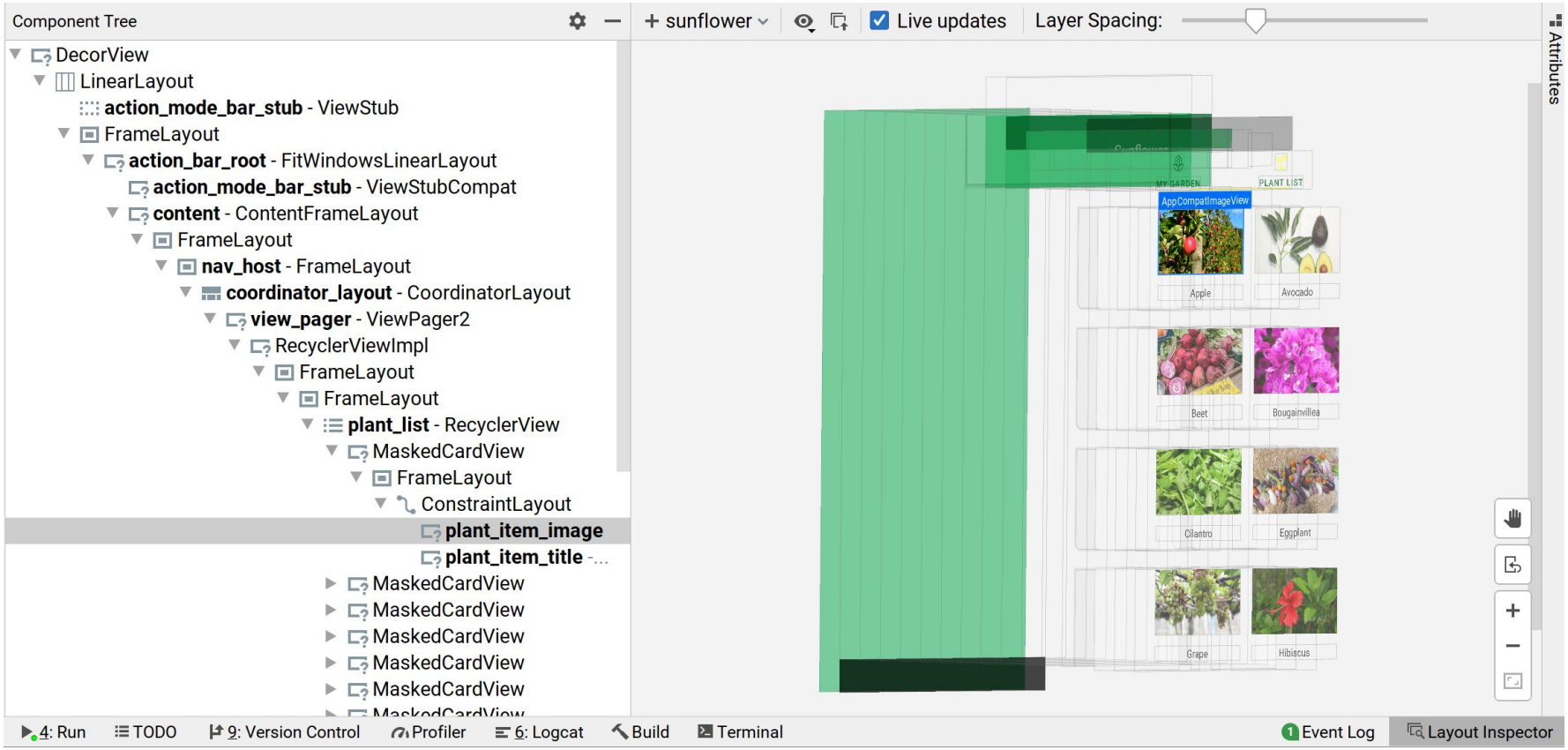
Platform Tools & Environments



1. Process
2. Profiling Sessions
3. Tools (Zoom, Time Selection)
4. Interaction Lane (Touches, Interactions)
5. Memory / Network Usage

Platform Tools & Environments

Android Layout Inspector



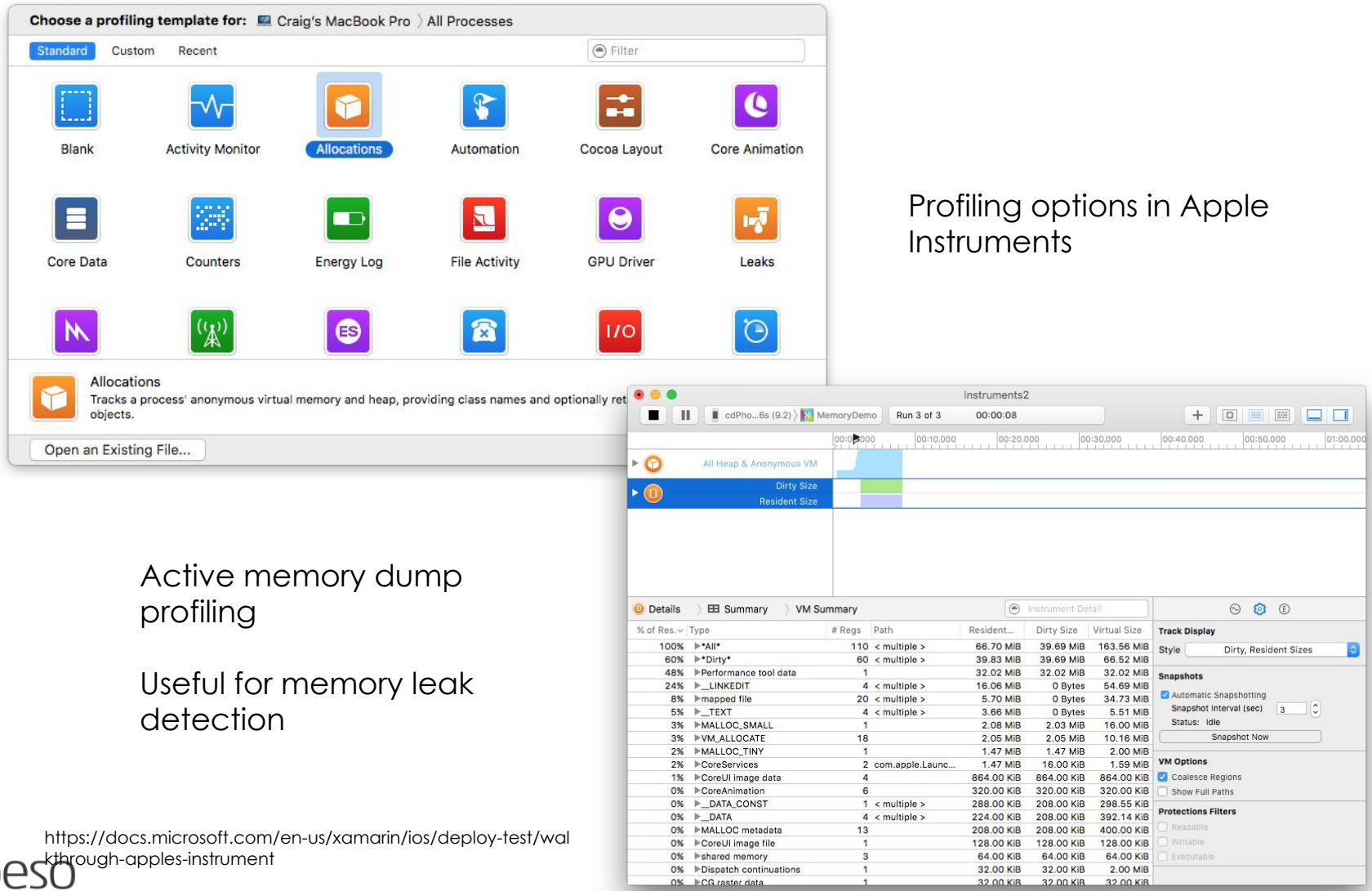
Platform Tools & Environments

- Tools provided in Software Development Kits (SDK)
- Android - Android Studio
  - Android SDK (currently Version 30)
  - Tools:
    - Android Debug Bridge (adb)
    - Systrace tool (systrace)
    - Fastboot (fastboot)
  - Android Profiler
  - Android Layout Inspector

- iOS - XCode
  - Plattform SDKs (iOS, MacOS, watchOS, tvOS)
  - Tools
    - xcodebuild (fastlane.tools)
    - xcrun (setup simulators)
    - xctest (test execution)
  - Apple Instruments (Performance analysis and testing tool)

Platform Tools & Environments

Apple Instruments



Profiling options in Apple Instruments

Active memory dump profiling

Useful for memory leak detection

https://docs.microsoft.com/en-us/xamarin/ios/deploy-test/walkthrough-apples-instrument

Emulator vs. Simulator

- Emulator
  - Mimic mobile device software, hardware and operating system
  - e.g. Android Emulator
  - Android Virtual Device (AVD) Manager

- Simulator
  - Mimics internal behavior of device, no hardware system
  - e.g. Apple Simulator (provided via XCode IDE)

Emulator vs. Simulator

Android Emulator

iOS Simulator

Test Software & Frameworks

Test Software & Frameworks

- Unit Testing - Running on development machine / device
  - **JUnit for Android** (Java & Kotlin)
  - **XCTest for iOS** (Swift)
    - **Nimble**

- Mocking
  - Simulate the behavior of complex, real objects
    - Android: Mockito-Kotlin, **MockK**
    - iOS: Mockit (no active development since 2019)

- UI Testing
  - **Appium (later today)**
  - Android
    - **Espresso (Developer UI Testing)**
    - UIAnimator (BlackBox UI Testing)
  - iOS
    - XCUITest (default)
    - Keep It Functional (KIF)

Android Unit Testing - without context

- Unit Testing “as you know”
- For helper methods, converts etc.

```
@RunWith(JUnit4::class)
class NoContextTest {

    @Test
     fun test_formatDateForUI_validData() {
        val date = LocalDate.now()
        val result = DateUtil.formatLocalDateForUI(date)
        assertEquals(
            DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG).format(date),
            result
        )
    }

}
```

Android Unit Testing - with context

- Unit Testing with classes having dependencies to Android context
- E.g. for testing methods accessing resources or testing binding adapters
- Code sample using Robolectric[1]

```
@RunWith(RobolectricTestRunner::class)
@Config(application = DemoTestApplication::class, sdk = [Build.VERSION_CODES.P])
class ContextTest {

    private lateinit var applicationContext: Context

    @Before
    fun setUp() {
        val app = ApplicationProvider.getApplicationContext() as DemoTestApplication
        applicationContext = app.applicationContext
    }

    @Test
    fun test_bindUsername_validUsername() {
        val tv = TextView(applicationContext)
        tv.bindUsername("Peter")
        assertEquals(String.format(applicationContext.getString(R.string.welcome_text), "Peter"),
           tv.text)
    }

}
```

[1] http://robolectric.org

Android Unit Testing - with mock data (1)

- For testing the independent method with predictable outcome
- E.g. for testing methods accessing external resources, testing methods of a “middle” layer
- Code sample using MockK[1]

```
@RunWith(JUnit4::class)
class MockTest {

    private lateinit var serviceLayer: ServiceLayer //The class to be tested

    @MockK
    lateinit var apiLayer: ApiLayer //The class which shall be mocked

    @Before
    fun setUp() {
        MockKAnnotations.init(this)
        mockkObject(apiLayer)
        serviceLayer = ServiceLayer(apiLayer)
    }

    @After
    fun tearDown() {
        unmockkAll()
    }

    ...
}
```

[1]https://mockk.io

Android Unit Testing - with mock data (2)

- For testing the independent method with predictable outcome
- E.g. for testing methods accessing external resources, testing methods of a “middle” layer
- Code sample using MockK[1]

```
...

@Test
fun test_useLastEmail_success() {
      every { apiLayer.getLoggedInUser() } returns User(“Peter”) // define what the method shall return
      val result = run { serviceLayer.getLoggedInUser() } // execute the method you want to test
      verify(exactly = 1) { apiLayer.getLoggedInUser() } // verify that the mocked method only got called
      once

      // assert result as usual
      assertNotNull(result)
      assertEquals(“Peter”. result.name)
}

...
```


[1]https://mockk.io

Android Unit Testing - MockK

- Mock void or returning methods
  - `every { apiLayer.getLoggedInUser() } just Runs` `// method without return`
  - `every { apiLayer.getLoggedInUser() } returns User("Peter")` `// method with return`
- Simply add co- before the method for suspend methods
  - `every { apiLayer.getLoggedInUser() } just Runs` `// normal method`
  - `verify(exactly = 1) { apiLayer.getLoggedInUser() }` `// normal method`
  - `coEvery { apiLayer.getLoggedInUser() } just Runs` `// suspend method`
  - `coVerify(exactly = 1) { apiLayer.getLoggedInUser() }` `// suspend method`
- Best Practice: Always specify number of calls in verify
  - `coVerify(atLeast = 1) { apiLayer.getLoggedInUser() }`
  - `coVerify(atMost = 1) { apiLayer.getLoggedInUser() }`
  - `coVerify(exactly = 1) { apiLayer.getLoggedInUser() }`
  - `coVerify(exactly = 1, inverse = true) { apiLayer.getLoggedInUser() }` `// everything BUT once`
- For running suspend methods add runBlocking (general Kotlin feature in CoroutineAwareTests)
  - `val result = repository.useLastEmail()` `// normal call`
  - `val result = runBlocking { repository.useLastEmail() }` `// blocking call`

Android UI Testing - Espresso (1)

- White Box UI Testing Framework provided by Google
- Tests mostly written by the developer
- System near and aware (e.g. waits automatically for app start)
- Basic behaviour follows three steps:
  - Find view
  - Perform action
  - Assert result

```
…

@Test
 fun test_enterSearchData_validData() {
    val searchTerm = “Clean Code”

    onView(withId(R.id.searchField))
               .perform(typeText(searchTerm),closeSoftKeyboard());

    onView(withId(R.id.searchButton))
                .perform(click());

    onView(withId(R.id.result))
                .check(matches(withText("Clean Code")));
}

…
```

Android UI Testing - Espresso (2)

- Lot of possible actions, commands, assertions…
  - withId() for accessing elements with an id
  - click() for clicking on elements
  - scrollTo() for scrolling to an element
  - matches() for comparing
  - isDisplayed() for checking if element is shown
  - isChecked() / isNotChecked() for checkboxes
  - isCompletelyDisplayed() for verifying if the full element is shown on screen

- Use cheatsheet for a good overview of options https://developer.android.com/training/testing/espresso/cheat-sheet

iOS Unit Testing

- Basic XCTestCase using XCTAsserts and Nimble

```
class TestClass: XCTestCase {

    func test_formateDateForUI_validData() {
        let date = Date()
        let result = DateUtil.formatDateForUI(date: date)

        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .medium

        /// Standard XCTAsserts
        XCTAssertNotNil(result)
        XCTAssertEqual(dateFormatter.string(from: date), result)

        /// Using Nimble
        expect(result).toNot(beNil())
        expect(result).to(equal(dateFormatter.string(from: date)))
    }
}
```

Mobile Test Automation

Mobile Test Automation

- Automate mobile device testing using specific test software

- Automation of UI Tests
  - Testing on different devices and emulators
  - Can be triggered on each build via CI / CD
  - Allows for testing many different scenarios automatically

- Example framework for Mobile Test Automation - Appium[1]:

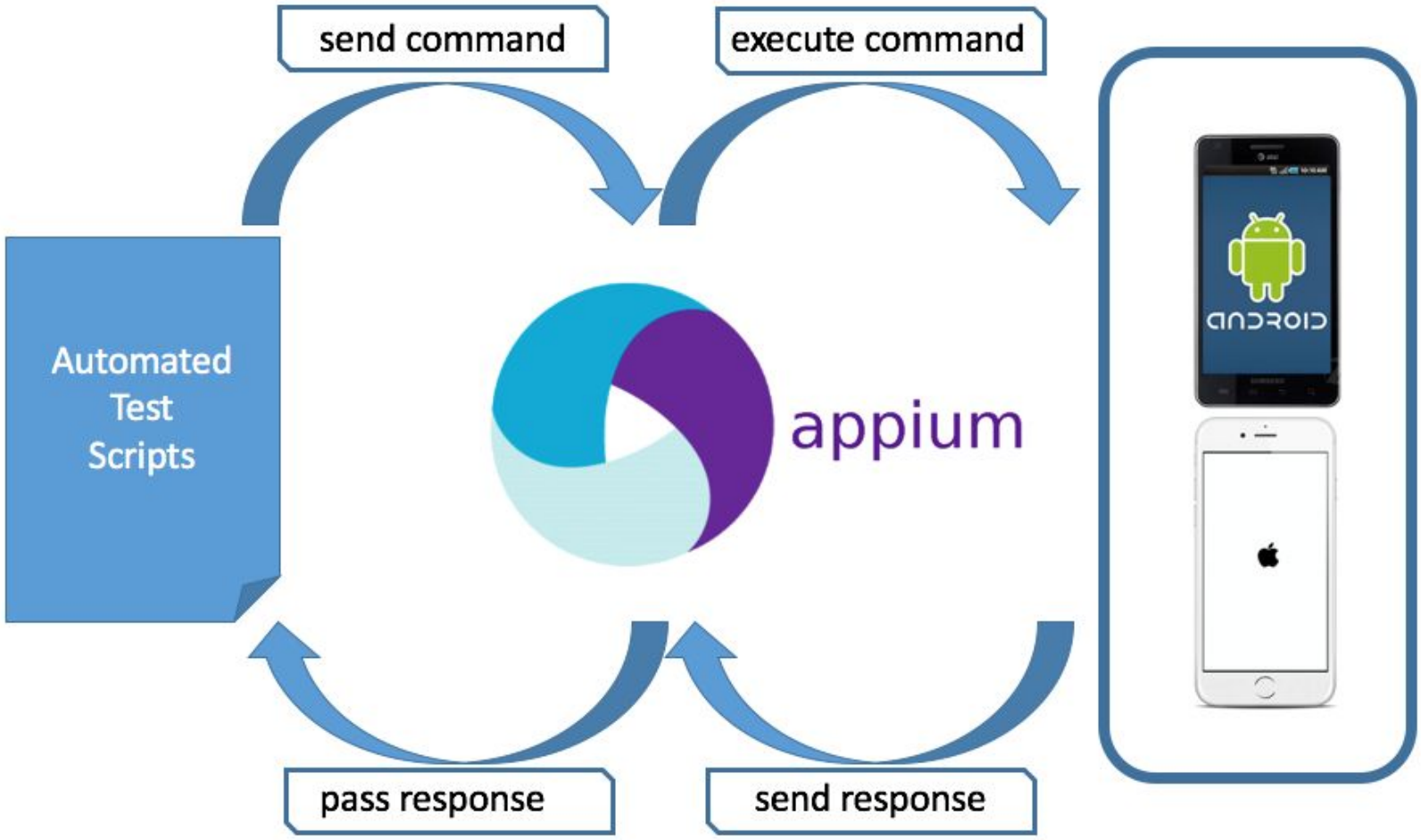
  - Open source automation tool
  - Allows for running test scripts on native applications, especially useful for mobile device
  - Platform agnostic -> Support for Android and iOS as well as web applications

[1]http://appium.io

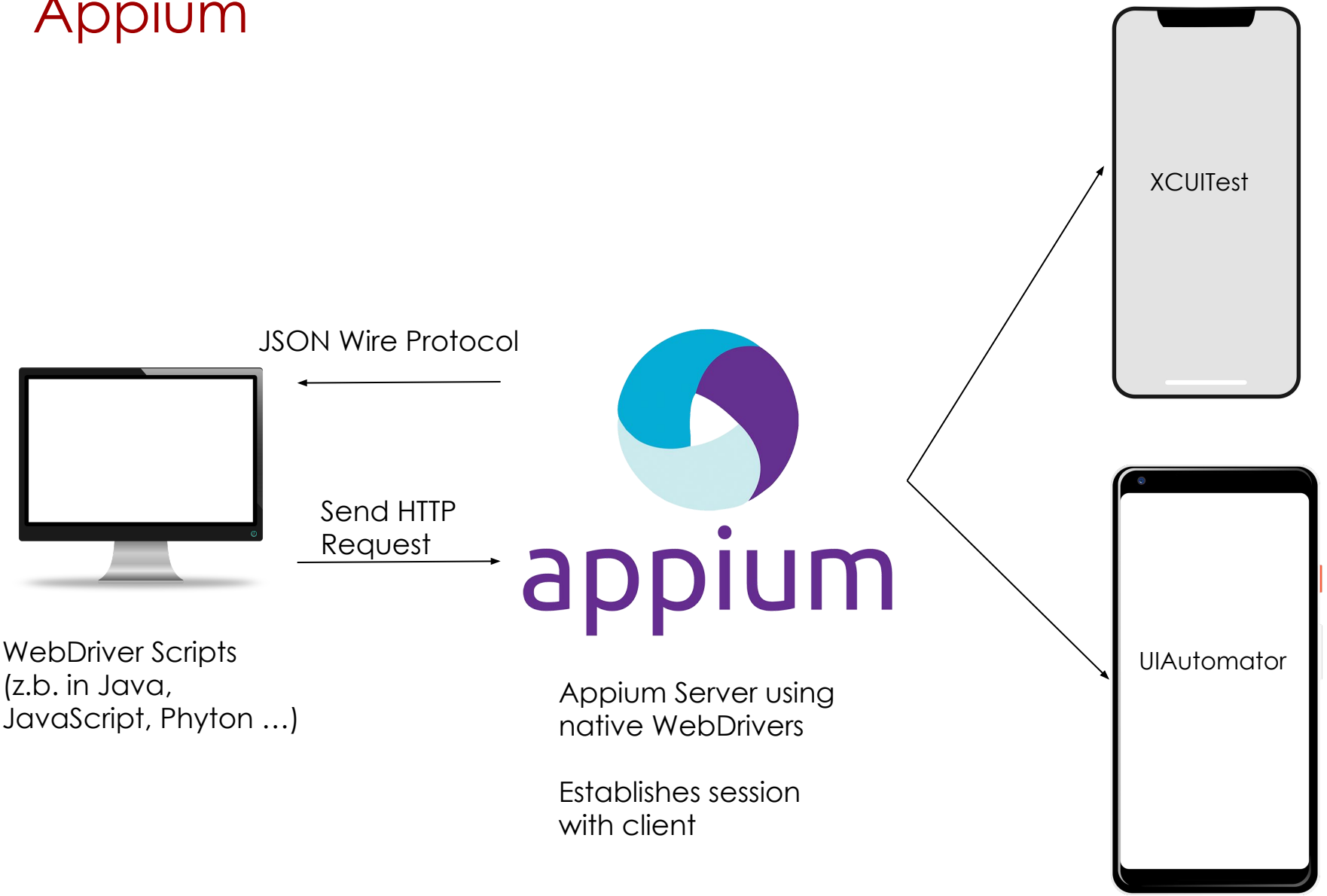
Appium Philosophy

- You shouldn't have to recompile your app or modify it in any way in order to automate it.
- You shouldn't be locked into a specific language or framework to write and run your tests.
- A mobile automation framework shouldn't reinvent the wheel when it comes to automation APIs.
- A mobile automation framework should be open source, in spirit and practice as well as in name!

Appium

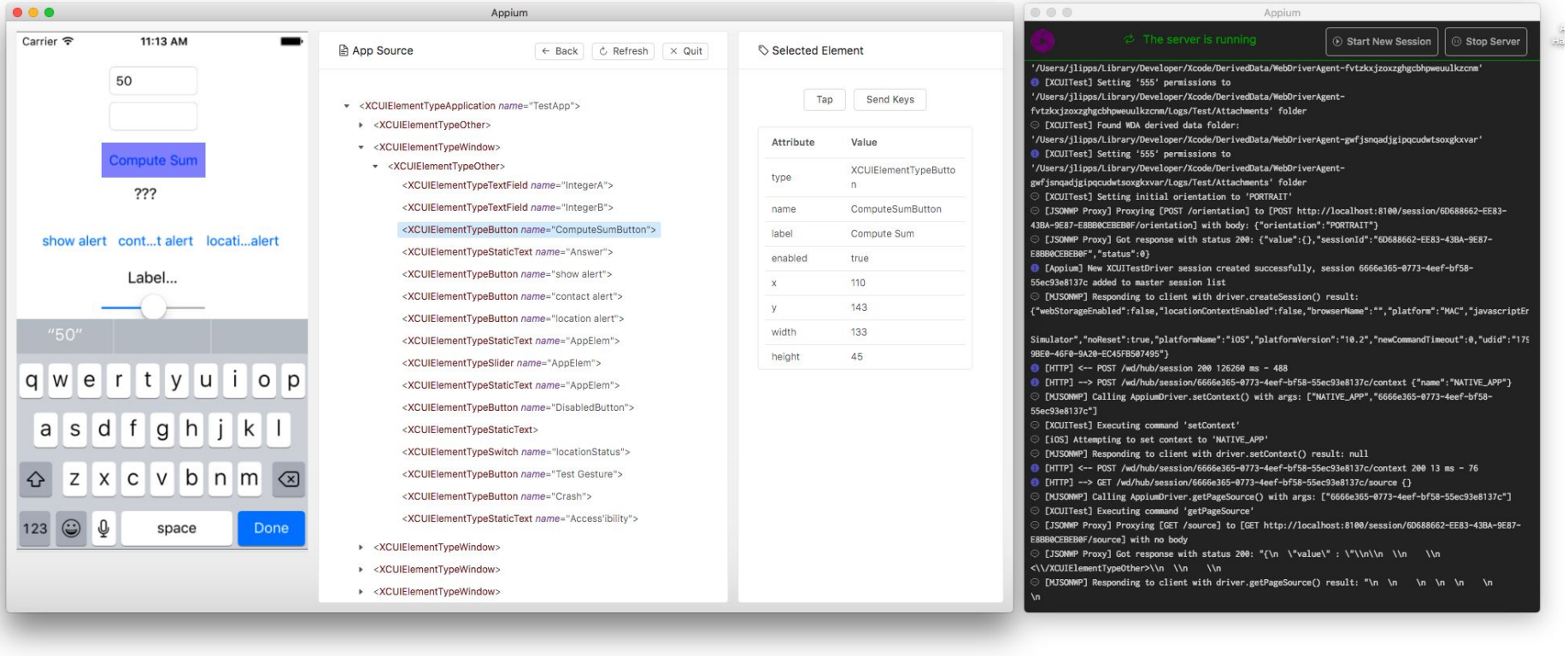

send command
execute command
Automated Test Scripts
appium
android
pass response
send response

Appium



For more Information, see
http://appium.io/docs/en/about-appium/getting-started/?lang=en

Appium Desktop



Cloud-based vs Local Mobile Testing

- Tests can be executed local or cloud-based
  - Using a on-premise solution and local devices
    - Knowledge and Experts needed to build test environment
    - High costs due to the fact that devices need to be bought
    - Generally considered more secure
  - Using a remote or cloud-based solution and devices
    - Easier setup
    - Wide variety of devices and configurations
    - Generally more cost effective
    - Service are for example:
      - e.g. kobiton.com - mobile device cloud platform
      - e.g motel.io - cloud based mobile test platform
      - e.g. mobileboxlab.com - cloud based device access provider
      - e.g. **Google Firebase Test Lab**[1]

[1]https://firebase.google.com/docs/test-lab

Google Firebase Test Lab

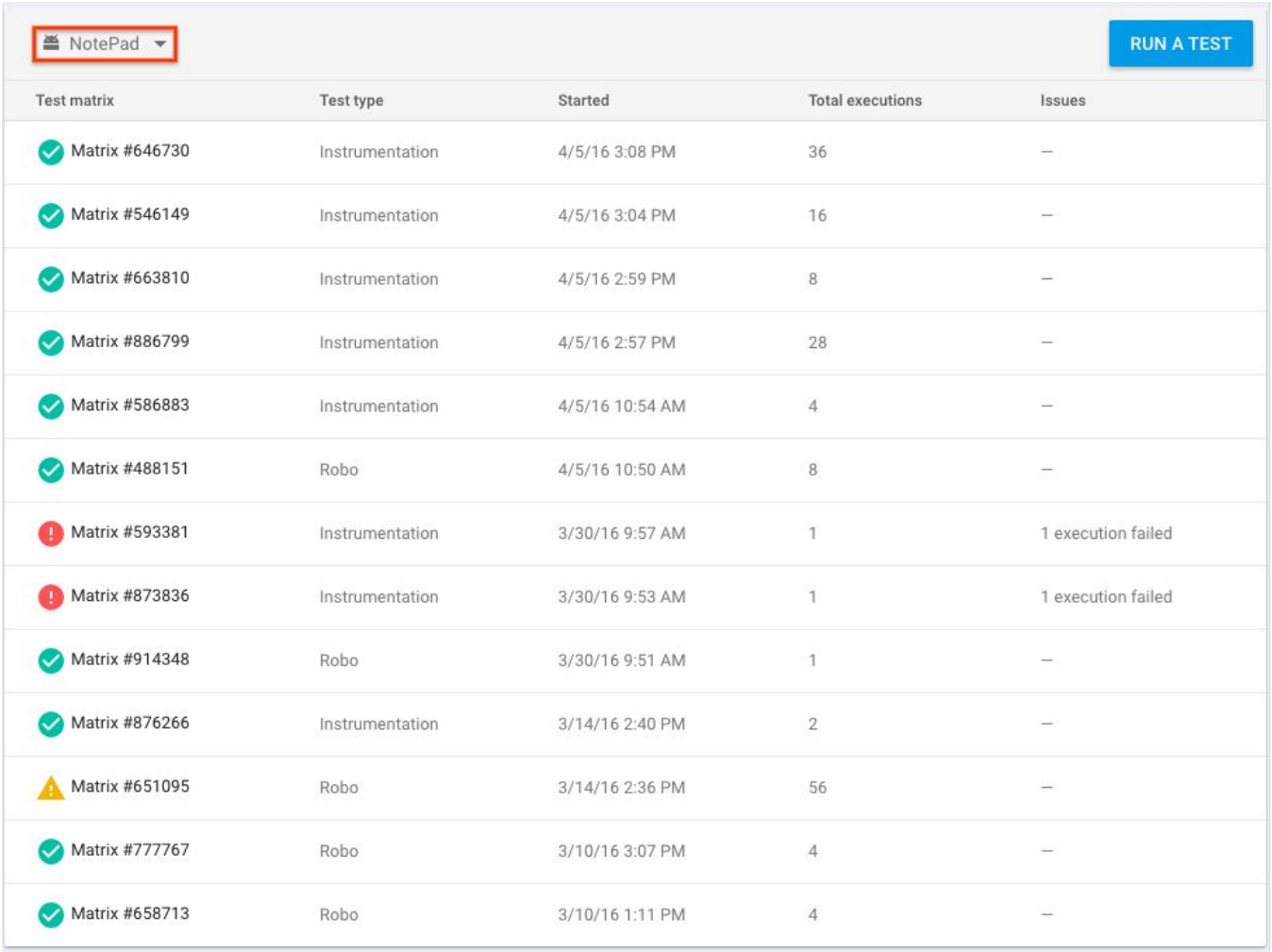
- Cloud Based App Testing Infrastructure
  - Multi-Platform support for Android & iOS
  - Wide variety of device
  - Usage of real devices (usually cheaper then buying them)
    - Available Devices see here
  - Allows for different device configurations

- For iOS, **XCTest** cases written by the developer are used
- For Android, **Espresso / UI Automator** test written by the developer are used
  - Or use **Robo Tests** which are auto-generated instrumentation tests by Google (Android only)

- Documentation: Run Android Tests on Firebase Testlab https://support.google.com/firebase/answer/6386654?authuser=0
- Example: Run Android Instrumationation Tests on Firebase Test Lab via CI/CD -> https://medium.com/firebase-developers/github-actions-firebase-test-lab-4bc830685a99

Google Firebase Test Lab

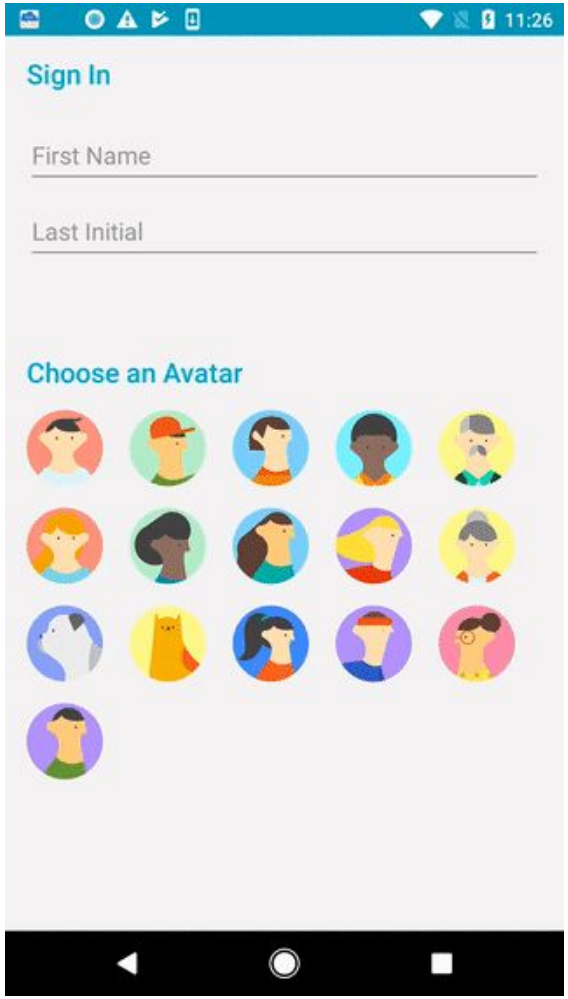
- Test Result Overview



Test matrix	Test type	Started	Total executions	Issues
Matrix #646730	Instrumentation	4/5/16 3:08 PM	36	–
Matrix #546149	Instrumentation	4/5/16 3:04 PM	16	–
Matrix #663810	Instrumentation	4/5/16 2:59 PM	8	–
Matrix #886799	Instrumentation	4/5/16 2:57 PM	28	–
Matrix #586883	Instrumentation	4/5/16 10:54 AM	4	–
Matrix #488151	Robo	4/5/16 10:50 AM	8	–
Matrix #593381	Instrumentation	3/30/16 9:57 AM	1	1 execution failed
Matrix #873836	Instrumentation	3/30/16 9:53 AM	1	1 execution failed
Matrix #914348	Robo	3/30/16 9:51 AM	1	–
Matrix #876266	Instrumentation	3/14/16 2:40 PM	2	–
Matrix #651095	Robo	3/14/16 2:36 PM	56	–
Matrix #777767	Robo	3/10/16 3:07 PM	4	–
Matrix #658713	Robo	3/10/16 1:11 PM	4	–

Google Firebase Test Lab

- Test Result Videos show execution of UI Tests (e.g. Robo Tests)



Google Firebase Test Lab

- Test Result Details & Error Logs

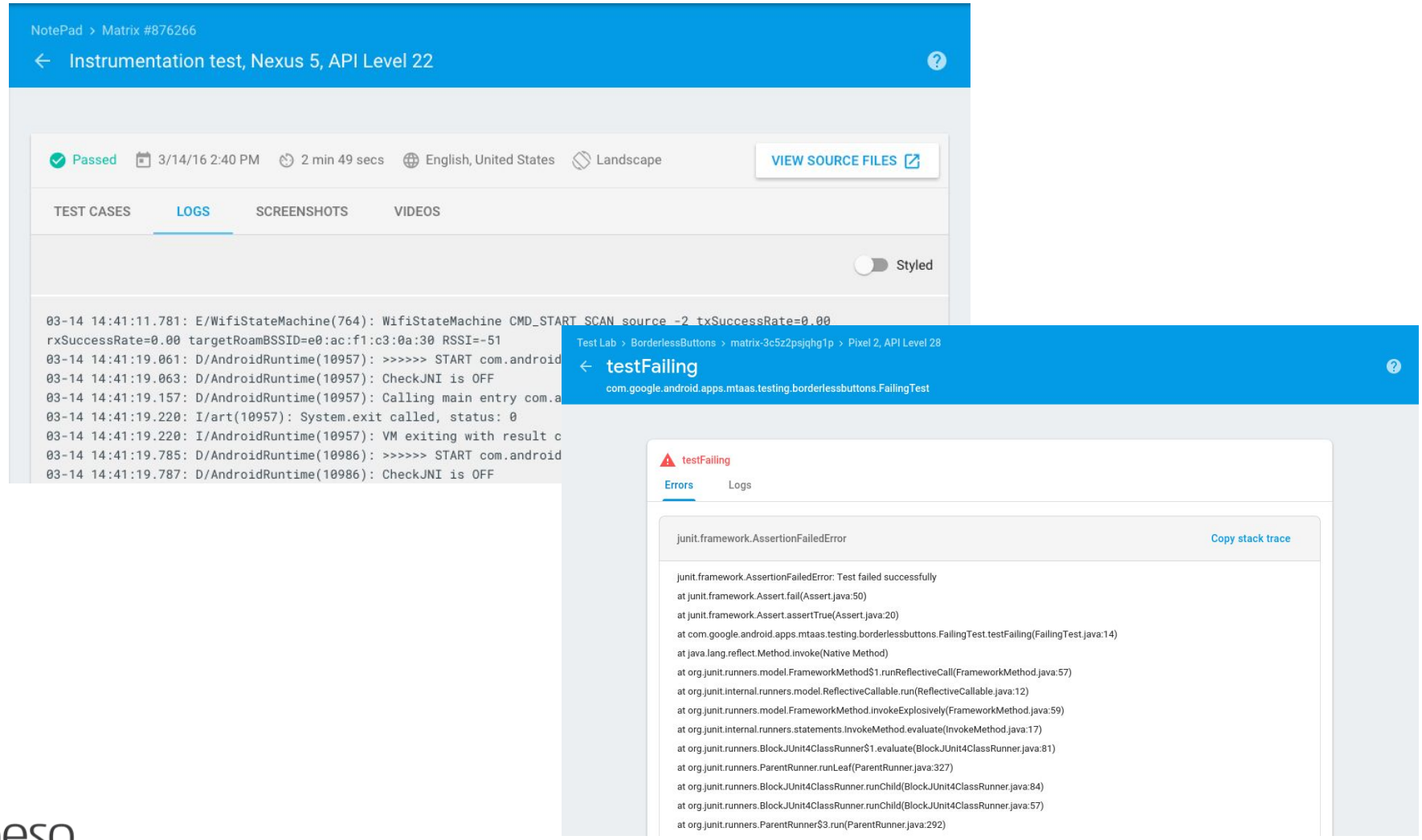


Image Sources

Image 1: iPhone screen sizes by deviceatlas.com (jkielty) -

`deviceatlas.com/blog/iphone-fragmentation-why-detecting-model-essential`

Image 2: Android profiler by Android Developers -

`developer.android.com/studio/profile/android-profiler`

Image 3: Appium overview by todaysoftmag.com:

`todaysoftmag.com/images/articles/tsm57/a11.png`

Image 4, 5, 6, 7: Test Results

`https://firebase.google.com/docs/test-lab/android/analyzing-results`

peso
PERFECTING SOFTWARE