**Mobile Testing**

**Apple-Iphone-IOS - apps are developed in C lang(41%)**

**Android-Samsung,Motorola…- Android – java lang(48%)**

**Blackberry**

**Windows**

**Mozilla Mobile OS**

**Native Apps** – developed in specific OS supported language(c, java) and needs that particular domain expertise

Fast, Reliable, best user experience and easy access to wider functionalities of device like camera, gps, contactList , accelerometer

Publishers can use Push notifications to alert for new things.

Eg: Facebook, linked in, Angrybirds, whatsApp etc….

**Disadv:**

More Costly – respective domain developers and pay them for each app

Enhancements need to be done for all platforms again..

**Hybrid Apps:**

Built in Html5, Css,JS(Like webapps) but they use some native code wrappers for wider functionalities like camera, gps etc…

Use plugins like phonegap/cordava &AppceleratorTitanium to wrap them into your specific operating system.

Easy to develop, Less costly, Enhancements can be done easily.

Eg: gmail, Uber, instagram, twitter

**Mobile WebApps:**

Load them using mobile browsers like safari, chrome

Slow depending upon on N/W connection

Look and feel is not that good when compared to your apps

Less features

Eg: Marriot, Mercedes-Benz ,whitebox-qa..etc…

**Mobile testing - types:**

* [**Usability testing**](http://www.softwaretestinghelp.com/usability-testing-guide/)– To make sure that the mobile app is easy to use and provides a satisfactory user experience to the customers
* Touch actions like swipe,pinch,zoom,scroll, tap….
* Orientations- horizontal/vertical- Landscape/portrait
* **Compatibility testing**– Testing of the application in different mobiles devices, browsers, screen sizes(Ipad,mobile devices with diff screen size,Tv) and OS versions according to the requirements – landscape/portrait modes..
* **Interface testing**– Testing of menu options, buttons, bookmarks, history, settings, and navigation flow of the application.
* **Services testing**– Testing the services of the application online and offline.
* **Low level resource testing**: Testing of memory usage, auto deletion of temporary files, local database growing issues known as low level resource testing.
* \*\*[**Performance testing**](http://www.softwaretestinghelp.com/introduction-to-performance-testing-loadrunner-training-tutorial-part-1/)– Testing the performance of the application by changing the connection from 2G, 3G,4G to WIFI, sharing the documents, battery consumption, etc.
* **Operational testing**– Testing of backups and recovery plan if battery goes down, or data loss while upgrading the application from store.
* [**Installation tests**](http://www.softwaretestinghelp.com/software-installationuninstallation-testing/)**–** Validation of the application by installing /uninstalling it on the devices.
* [**Security Testing**](http://www.softwaretestinghelp.com/category/security-testing/)– Testing an application to validate if the information system protects data or not.

[**https://en.wikipedia.org/wiki/Mobile\_application\_testing**](https://en.wikipedia.org/wiki/Mobile_application_testing)

**Mobile Testing -Automation**:

**Why:**

Android suffer more **fragmentation** than IOS, many variants w.r.to OS ,Devices - still we need high quality, robust and reliable app - and so we need to do thorough testing in less time and competitive pressure(Test apps are developed in agile-2 weeks sprints) - so we need automation by following **cross-platform** method.

\*\*Agile cycle – developers(develop code, fix bugs &checkin)-Testers(get updated url/code & test)

Automation enables simultaneous testing on hundreds of real devices. It increases efficiency & makes things faster i.e., app is brought soon to market.

In other words automation is **Cost-effective** method for regression testing of software products that have a long maintenance life - save time and money with same resources of manual testing.

**Main Challenges of Mobile Testing:**

<http://www.slideshare.net/ElizabethHechtel/mainchallengesofmobiletesting2compressed>

**Different ways to do automation:**

Coding test scripts - best way though it takes time

Record and playback -tools

Automatic test exercises- for quick smoke testing - not that good coverage.

**Key focus areas as QA:**

**UI layouts and elements**

Games especially are typically targeted at a high number of different **screen resolutions and screen types**. Regression testing should be done each and every time the UI’s layout changes to ensure that the game works.

**Menu structure and functions**

Testing menu structures, functionality and behavior can be automated with instrumentation and the help of different test-automation frameworks.

**Screen orientation**

Surprisingly, so many apps and games out there get this wrong. If a screen’s orientation changes during an interaction, for example, what happens? What is supposed to happen? Does the app or game work well in both landscape and portrait modes?

**Screen resolution**

A lot of screen resolutions exist, especially on Android, and auto-scaling will usually help developers. However, test your game across these resolutions to ensure that the graphics do not stretch.

**Performance** - under load and stress tests, Testing performance is critical because this will make make or break the user experience. Performance lag, for example, is easy to expose with real devices.

**User interaction and responsiveness** - usabilty & navigation flow

**Background events**

Interruptions, battery consumption and the effect of battery chargers on overall performance and usage all have a significant impact on the user experience — and entertainment value.

**Muti User Features**

Nowadays, multi-user support is common in both apps and games. Testing multi-player capabilities is important and is naturally more challenging

**Social integration – F/B, twitter**

**Securities & Liabilities** - 3rd party licenses

**Different mode of entry –** landing to same page from different places should still work. Eg: inbox functionality of gmail is being called from inbox click and also AllMail..

**Emulator and Simulator:**

**Emulator – duplicate every aspect of mobile including H/W**

**Simulator- similar to device but some programs may run differently**

**An emulator** is an application that emulates real mobile device software, hardware and operating systems, allowing us to test and debug our applications.

Emulators are more suitable for debugging. Often an emulator comes as a complete re-implementation of the original software.

**Emulator is usually provided by device manufacturer**. Emulators are written in machine-level assembly languages.

eg: Android (SDK) Emulator , Windows phone emulator

**A simulator** may be created by the device manufacturer or by some other company. Simulators are written in high level languages.

A simulator is a **less complex application** that simulates internal behavior of a device, but does not emulate hardware and does not work over the real operating system.

Simulators can be difficult for debugging purpose. Simulator is just a partial re-implementation of the original software .

e.g. - iOS Simulator , Blackberry simulator

<http://www.slideshare.net/videotuition/mobile-application-testing-emulators-vs-simulators-real-mobile-device-comparison-usage>

**Limitations of Mobile Simulators and Emulators:**

Emulators can’t completely mimic a given device.

As for performance, an emulator may underperform or outperform the actual mobile device. eg: running the emulator on a laptop with a 2.4 GHz processor isn’t going to flush out the performance issues on a mobile device equipped with a 600 GHz processor.

Emulators are useful for prototyping and uncovering problems early on in the app design and development, testing lifecycle. They can give developers a quick take on whether an app will run on a particular device, without having to initially invest in that device.

So verification / hardware testing is required at later point of time

**Different Test Automation Frameworks:**

**Robotium** is an Android test automation framework that fully supports native and hybrid applications. Robotium makes it easy to write powerful and robust automatic black-box UI tests for Android applications. With the support of Robotium, test case developers can write function, system and user acceptance test scenarios, spanning multiple Android activities.

**Uiautomator** - by Google, provides an efficient way to test UIs. It creates automated functional test cases that can be executed against apps on real Android devices and emulators. It includes a viewer, which is a GUI tool to scan and analyze the UI components of an Android app.

**Espresso** - by Google, is a pretty new test automation framework that got open-sourced just last year, making it available for developers and testers to hammer out their UIs. Espresso has an API that is small, predictable, and easy to learn and built on top of the Android instrumentation framework. You can quickly write concise and reliable Android UI tests with it.

**Calabash** is a cross-platform test automation framework for **Android and iOS** native and hybrid applications. Calabash’s easy-to-understand syntax enables even non-technical people to create and execute automated acceptance tests for apps on both of these mobile platforms.

**Selendroid** – apps developed using Android SDK version<16

**Appium**—from **Saucelabs-** is a open source mobile test **automation Cross platform mobile automation tool/fw** – **Android, IOS etc.**

It is http server in node.js –which exposes REST API when appium server is setup in our system.

Request and responses are carried out using JSonWireProtocol framework (and tool) for **native, hybrid and mobile-web.**

In its architecture, Appium is an HTTP server written in Node.js that creates and handles multiple WebDriver sessions.

Appium starts tests on the device and listens for commands from the main Appium server.

Appium supports almost all languages such as Java, Objective-C, JavaScript, PHP, Ruby, Python or C#, among many others

\*\*Also w.r.to mobile games - Appium’s test automation helps to build **image recognition** into the test-automation process.

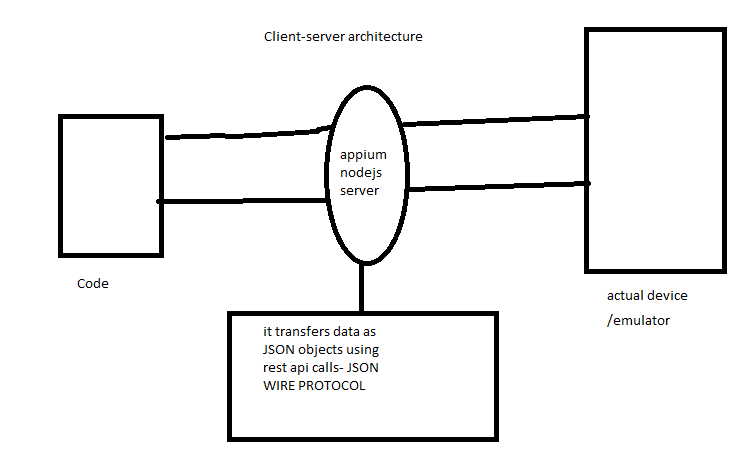
Testing can be done using device or emulator or simulator.

Each vendor like IOS, android have different way of running test case on device- using appium these executions are handled by f/ws like UIAutomator(android), InstrumentsController (IOS).

UIAutomator is androids native UI automation F/W which supports running unit test cases(written in any language like java,RubyonRails etc) directly to device from cmd line.

IOS-appium inspector.

**Appium Architecture**: This follows client server architecture where client is WebDriver code and server is any device.

****

**Java-JDK, Andoid SDK(Android native apps), IOS(xcode), Appium.exe or nodejs,**

**Appium-client specific (appium java client jar), to test any app - apk(for android code bundling)/ipa file(source code bundled into a package for ios apps)**

**eclipse is used for java development**

**AndroidStudio is an IDE used for android apps development**

**Xcode is and IDE used for IOS apps development**

**Appium Setup:**

1. Download Java-jdk

setup environment variable - JAVA\_HOME- C:\Program Files\Java\jdk1.8.0\_45

path – append to existing path - %JAVA\_HOME%\bin

[**http://www.oracle.com/technetwork/java/javase/downloads/index.html**](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

#### 2. Download Android SDK -https://developer.android.com/studio/index.html – navigate to Get just the command line tools – download specific package

#### Last step of your installation it open SDK manager view(either fom folder or from AllApps) and select tools, platform-tools, build-tools, also u can select android version required to test – click on install packages.

#### Environment variables:

#### ANDROID\_HOME - C:\Users\Training room\AppData\Local\Android\android-sdk

#### Path variable:

#### Add %ANDROID\_HOME%\tools

#### Add %ANDROID\_HOME%\platform-tools

Optional : install ADT(android dev) plugin in eclipse from help-> marketplace – this helps to view Logcat(for logs) and open DDMS(Debugging monitor service) perspective.

To cross check – type android in cmd prompt and see it opens your SDK manager view.

#### 3. Download node.js - <https://nodejs.org/en/download/>

#### (or)

#### Download Appium for Windows zip archive and extract all files in a folder called Appium - <http://appium.io/> - download appium

#### Run Appium client from folder or your all apps & click on start server icon.

#### 4.

#### Android native app testing:

#### Create device using AVD manger from Android installation. We created emulator but we can also connect real devices (download pdanet app on to your laptop/PC and also mobile phone)

#### You need either AVD manager created emulator or real device

#### Start device by clicking on start in avd manager and run adb devices cmd from your cmd prompt to see whether device is connected and its ready.

#### IOS native app testing:

#### You can skip step 2- android sdk installation is only required for android apps testing

#### Instead of android sdk – you can download xcode(IDE for IOS apps development) xcode will be used to create device simulator

#### If you are using real device, then you can directly connect the device

#### Authorize the device

#### UDID

#### 5. we need .apk file for testing or if we have source code we can create apk file from androidstudio(ide for android apps development)

#### For testing purpose – you can Get any .apk file from google.com for app you want to test and give that file location in your code

#### In real time projects – development team gives zip version of apk file to be used or we generate the file from source code using appropriate IDE’s

#### .apk file is bundled or packaged android code used for distribution/ installation of mobileapps.

#### For ios apps – we need .ipa file for testing or we can get source code and bundle it using xcode.

#### ipa file is bundled or packaged ios(objective C) code used for distribution/ installation of mobileapps.

#### 6. Download appium java client jar

If its maven project- just add <dependency>

<groupId>io.appium</groupId>

<artifactId>java-client</artifactId>

<version>3.4.0</version>

#### </dependency>

### **Appium Concepts**

**Client/Server Architecture**  
Appium is at its heart a webserver that exposes a REST API. It receives connections from a client, listens for commands, executes those commands on a mobile device, and responds with an HTTP response representing the result of the command execution. The fact that we have a client/server architecture opens up a lot of possibilities: we can write our test code in any language that has a http client API, but it is easier to use one of the [Appium client libraries](http://appium.io/downloads). We can put the server on a different machine than our tests are running on. We can write test code and rely on a cloud service like[Sauce Labs](https://saucelabs.com/mobile) to receive and interpret the commands.

**Session**  
Automation

(i.e., a map or hash) sent to the Appium server to tell the server what kind of is always performed in the context of a session. Clients initiate a session with a server in ways specific to each library, but they all end up sending a POST /session request to the server, with a JSON object called the 'desired capabilities' object. At this point the server will start up the automation session and respond with a session ID which is used for sending further commands.

**Desired Capabilities**  
Desired capabilities are a set of keys and values automation session we're interested in starting up. There are also various capabilities which can modify the behavior of the server during automation. For example, we might set the platformName capability to iOS to tell Appium that we want an iOS session, rather than an Android one. Or we might set the safariAllowPopups capability to true in order to ensure that, during a Safari automation session, we're allowed to use JavaScript to open up new windows. See the [capabilities doc](http://appium.io/slate/en/master/#caps.md) for the complete list of capabilities available for Appium.

**Appium Server**  
Appium is a server written in Node.js. It can be built and installed [from source](http://appium.io/slate/en/master/#appium-from-source.md) or installed directly from NPM:$ npm install -g appium $ appium

**Appium Clients**  
There are client libraries (in Java, Ruby, Python, PHP, JavaScript, and C#) which support Appium's extensions to the WebDriver protocol. When using Appium, you want to use these client libraries instead of your regular WebDriver client. You can view the full list of libraries [here](http://appium.io/slate/en/master/#appium-clients.md).

[**Appium.app**](https://github.com/appium/appium-dot-app)**,**[**Appium.exe**](https://github.com/appium/appium-dot-exe)  
There exist GUI wrappers around the Appium server that can be downloaded. These come bundled with everything required to run the Appium server, so you don't need to worry about Node. They also come with an Inspector, which enables you to check out the hierarchy of your app. This can come in handy when writing tests.

**Appium Platform Support**

Appium supports a variety of platforms and testing modalities (native, hybrid, web, real devices, simulators, etc…). This document is designed to make explicit the level of support and requirements for each of these.

**IOS SUPPORT**

See Running on OS X: iOS for iOS requirements and setup instructions.

Versions: 7.1, 8.0, 8.1, 8.2, 8.3, 8.4, 9.0, 9.1 and 9.2

Devices: iPhone Simulator, iPad Simulator, and real iPhones and iPads

**Native app support:** Yes, with debug version of .app (simulator), or correctly-signed .ipa (real devices). Underlying support is provided by Apple’s UIAutomation framework.

**Mobile web support**: Yes, via automation of mobile Safari. For real devices, ios-webkit-remote-debugger is required, and automation of native aspects of the Safari interface is not possible. See the mobile web doc for instructions.

**Hybrid support:** Yes. For real devices, ios-webkit-remote-debugger is required. See the hybrid doc for instructions.

Support for automating multiple apps in one session: No

Support for automating multiple devices simultaneously: No

Support for automating vendor-provided or third-party apps: Only vendor-provided apps (Preferences, Maps, etc…), and only on the simulator

Support for automating custom, non-standard UI controls: Minimal. You need to set accessibility information on the control which enables some basic automation.

**ANDROID SUPPORT**

See Running on OS X: Android, Running on Windows, or Running on Linux for Android requirements and setup instructions.

Versions: 2.3 and up

Versions 2.3 through 4.2 are supported via Appium’s bundled version of Selendroid, which utilizes Instrumentation. Selendroid has a different set of commands than the default Appium (though this is rapidly being minimized) and a different support profile. To access this automation backend, use the automationName capability with the value Selendroid.

Versions 4.2 and up are supported via Appium’s own UiAutomator libraries. This is the default automation backend.

Devices: Android emulators and real Android devices

Native app support: Yes

Mobile web support: Yes (but not when using Selendroid backend). Automation is effected using a bundled Chromedriver server as a proxy. With 4.2 and 4.3, automation works on official Chrome browser or Chromium only. With 4.4+, automation also works on the built-in “Browser” app. Chrome/Chromium/Browser must already be installed on the device under test. See the mobile web doc for instructions.

Hybrid support: Yes. See the hybrid doc for instructions.

With default Appium automation backend: versions 4.4 and up

With Selendroid automation backend: versions 2.3 and up

Support for automating multiple apps in one session: Yes (but not when using the Selendroid backend)

Support for automating multiple devices simultaneously: Yes, though Appium must be started using different ports for the server parameters --port, --bootstrap-port (or --selendroid-port) and/or --chromedriver-port. See the server args doc for more information on these parameters.

Support for automating vendor-provided or third-party apps: Yes (but not when using the Selendroid backend)

Support for automating custom, non-standard UI controls: No

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write your testscripts with driver configurations and DesiredCapabilties(details about your device and OS configuration) and apk file location.

Set app to be tested and launch its home page using your code:

capabilities.setCapability(“appPackage”,

capabilities.setCapability(“appActivity”,

How to get “appPackage” & activity from apk file:

**1. using appium client--choose application path in apiium ui-android settings(or)**

**2. copy to C:\Users\Training room\AppData\Local\Android\android-sdk\build-tools\23.0.3 and go to cmd prompt here & run aapt dump badging yourapkfilename.apk**

3. connect actual device & open app to be tested & run cmds

After these 6 steps are done run your code finally – initially u can run once which installs ur app in emulator for first time when u run code. ☺Then again to locate elements use UIAutomator as below

\*\*\*\*\*\*\*\*

**ADB**-android debug bridge - Android >> android-sdk installation folder>> platform-tools >> adb:

adb logcat

adb start-server

adb kill-server

**To Locate/inspect elements:** Use UIautomateViewer Or appium Inspector for android devices.

You can use appium inspector for IOS devices

C:\Users\Training room\AppData\Local\Android\android-sdk\tools -> uiautomatorviewer.bat

After this go to emulator and click on ur app to test and go to screen u want to automate. Then click on second left button of Uiautomator to take screenshot of page.

Once the Screenshot of the device is taken and displayed on the window of UIAutomator, click on Save button.

It will open up the small window which will look like the below image.

How to Open the UI Dump files:

After saving all the screenshots of the different screens of your application, it is the time to see how to open the desired screenshot.

1) Click on the Open button of the UIAutomator window. A small window will open up.

2) Now browse the Screenshot and the UiX file and click on OK button.

\*\*\*\*\*\*\*\*\*\*\*\*

UIAutomatorViewer for android native apps, does not help us in inspecting elements which are under WebView.

Appium inspector- for IOS

**Finding elements:**

The difference between Find Element and Find Elements method is the first returns a WebElement object otherwise it throws an exception and the latter returns a list of WebElements, it can return an empty list if no DOM elements match the query.

**Content-desc or text fields** - can be used exactly like text attribute with **findElementByName**.

**Class** - **findElementByClassName**

driver.findElementByClassName("android.widget.ImageView").click();

//or

driver.findElement(By.className("android.widget.ImageView")).click();

**resource id**- most efficient way - **findElementById**

\*\*\*ByXpath- driver.findElementByXPath("//android.widget.ScrollView [@id= sign\_up\_screen]/following-sibling::android.view.View/android.widget.EditText").sendKeys("test");

\*\*FIND ELEMENT USING PARENT NODE:

WebElement parentElement = driver.findElement(By.name("Amazon Sign In"));

List<WebElement> childElements = parentElement.findElements(By.className("android.view.View"));

WebElement mainElement = childElements.get(4);

mainElement.findElement(By.className("android.widget.EditText")).sendKeys("Your\_USerName");

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chrome DevTools helps us in following:

Debugging websites in browser tabs.

Debugging WebViews in native Android apps.

Screencasting live to your development machine from your Android device.

**Real time Challenges with appium:**

Configuring devices- with variant OS and devices in market -Must have good device knowledge to configure many devices in AVD manager.

Locating elements - Need to write your xpath to locate element uniquely sometimes when u don’t have any unique id or name.

--Code do not run if configuration fails - Need to do proper configurations and get through errors by understanding log messages.

Lack of System memory - to run your tests across fifteen or twenty devices

You can add on if any☺

\*\*\*\*

IOS apps using appium:

<http://appium.io/slate/en/tutorial/ios.html>

android specific….

adb: AndroidDebugBridge – It’s a cmd line tool in android sdk, it helps to communicate with device using the cmds

ddms- logcat, screen capture ,…

logcat: can view the application logs for debugginf purpose

 For [automating mobile gestures](http://appium.io/slate/en/master/#touch-actions.md) like tap, flick, and swipe :

Certain driver methods are avialble and we also use TouchAction class

<http://appium.io/slate/en/master/?java#automating-mobile-gestures>

…………..

Grid:parallel testing

<http://appium.io/slate/en/master/?java#selenium-grid>

references:

http://appium.io/getting-started.html?lang=en

Appium- client server architecture, it is basically server based on node.js

Philosophy:

1. You should be using the same automation language you are familiar for web(java,ruby,python,C#).
2. You should not change or add any scripts on the app under test to perform

Automation.

1. It should be opensource

Appium wraps internally Vendor provided libraries to perform automation

Android- UIAutomator

IOS-UIAUtomation

It based on WebDriver..

UIAutomator- for testing android apps—It is API or testing fw given by android to test the features and functionalties of app.

<https://developer.android.com/topic/libraries/testing-support-library/index.html#UIAutomator>

two ways to setup apppium server:

1. using node npm packages
2. using gui exe files

android sdk- its development tool kit for android developers just like jdk for java developers

**The HTTP request carries this language preference information in the Accept-Language header.**

**It is typically a two- or three-letter language code (eg. fr for French), followed by optional subcodes representing such things as country (eg. fr-CA represents French as used in Canada**).

If the server is set up to return alternate versions of a page or resource in more than one language, and the language you prefer is available, it will go through a process known as HTTP content negotiation to find content in the language(s) indicated in the HTTP request. If there is only one version of a page on the server, that version will be retrieved. If none of the languages you request are available, the server should be set up to return a default language choice.

Most mainstream browsers allow you to modify the language preferences that your browser sends with the HTTP request. The value itself should conform to BCP 47 (Tags for the Identification of Languages). **It is typically a two- or three-letter language code (eg. fr for French), followed by optional subcodes representing such things as country (eg. fr-CA represents French as used in Canada**). Regions can include larger areas than countries. If you set Spanish for Latin America, you are likely to set es-419. (Learn more about language tags.)