How Do I Get Started with Mobile Testing?

Theodore Chan

TheodoreEChan@gmail.com

Abstract

With over 60% of U.S. mobile subscribers using smartphones, writing and testing high quality mobile applications that provide user value is critical. Users will uninstall your app if it's not delivering a great user experience especially if it has crashes, bugs, performs poorly, drains the battery excessively or takes too long to start..

This paper is aimed at the beginner mobile tester to assist you in developing a mobile testing strategy; implement a testing process; and iterating to a testing system with a mixture of manual and automated testing for iOS and Android. This paper will guide you as your begin your mobile testing adventure, and provide pointers as you progress through your own testing journey.

Biography

Theodore Chan is a Software Development Engineering in Test (SDET) – Senior Quality Engineer at Wellero, Inc., in Portland, Oregon where he focuses on Mobile Testing and Mobile Automation. He is passionate about being an advocate for the user, software quality, process improvements and finding creative ways to improve user experiences.

Theodore has a B.S. in Computer Engineering from Rose-Hulman Institute of Technology. He is a Certified ScrumMaster (CSM) and Certified Scrum Product Owner (CSPO).

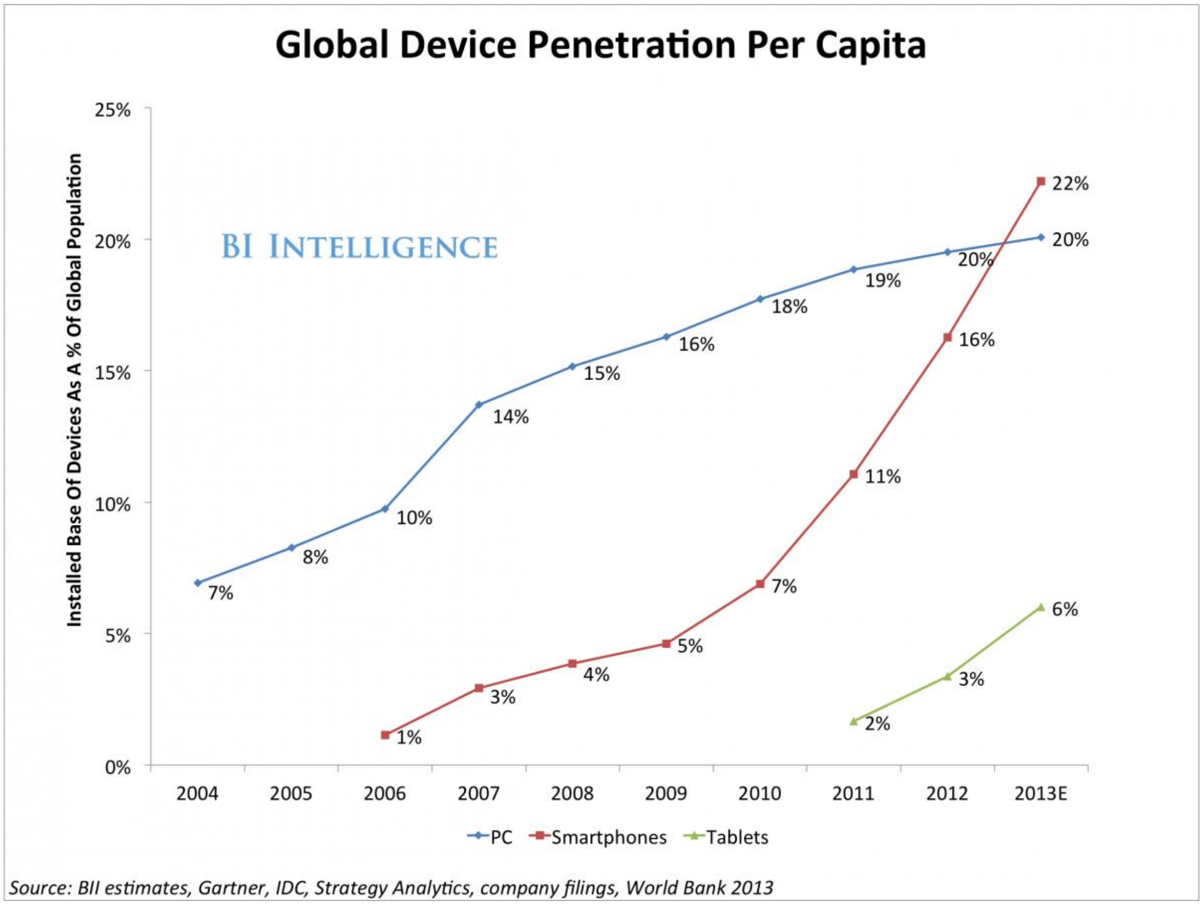
1 Introduction

This paper provides the practical tips for beginning with mobile testing and walks through some of the experiences, challenges, pitfalls and successes of mobile application testing. At the end of this paper the testing professional will be able to begin their own mobile testing journey incorporating what they know from traditional testing and applying it to the new and exciting space of mobile testing.

Why should software developers and testers care about mobile?

Mobile Phones, Smartphones, Mobile Apps and Mobile Testing are all hot topics in Software Development. In comScore's Reports January 2014 Subscriber Market Share, 159.8 million people in the United States or 66.8% of the mobile market owned smartphones.

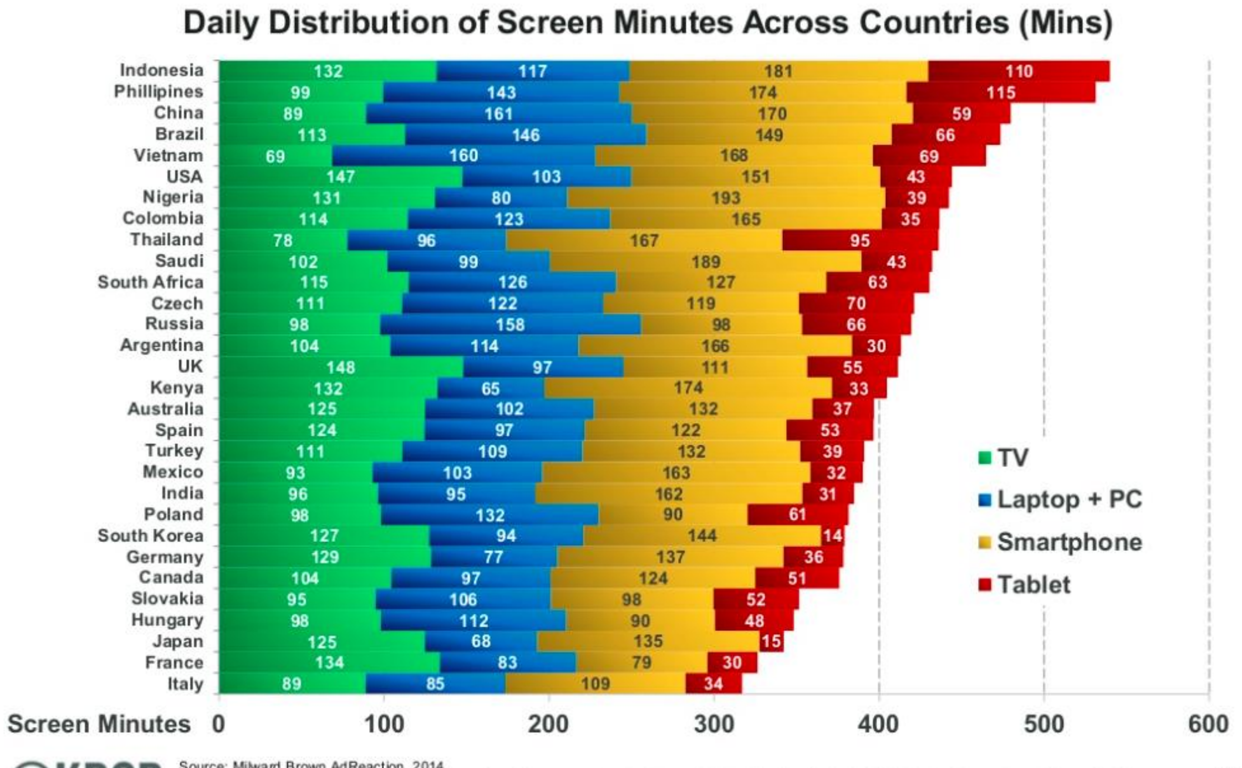
Looking at statistics worldwide, according to a Business Insider's article, "One In Every 5 People In The World Own A Smartphone, One In Every 17 Own A Tablet", by the end of 2013 22% of the global population will own a smartphone. "On average, there will be two smartphones for every nine people on earth, or 1.4 billion smartphones, by the end of 2013." eMarketer projects that by the end of 2014, the global smartphone audience will total 1.75 billion or one in four people of the world population.



Source: BII estimates, Gartner, IDC, Strategy Analytics, company filings, World Bank 2013

http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

Sometime in mid-2012 smartphones surpassed PC ownership worldwide. PC ownership has levelled off, while smartphone and tablets continue exponential growth. For some people, the smartphone may be the only access that they have to the internet.



Source: Milward Brown AdReaction, 2014.
Note: Survey asked respondents "Roughly how long did you spend yesterday watching television (not online) / using the internet on a laptop or PC / on a smartphone or tablet?" Survey respondents were age 16-44 across 30 countries who owned or had access to a TV and a smartphone and/or tablet. The population of the 30 countries surveyed in the study collectively represent ~70% of the world population.

http://www.kpcb.com/internet-trends

In Mary Meeker's, "Internet Trends 2014 - Code Conference", Smartphones are a primary screen over TV and Laptop + PC for a high percentage of the countries in world.

Software quality developers and testers are moving to where the users are spending their time and adapting to smartphones and tablets for their applications. They want to meet the needs of their users, adapt their current product lines and build new and innovative solutions for their users.

2 Mobile Testing Background

Mobile Testing has many close ties to desktop, traditional web app, and UI testing; however there are many more variables that can throw a wrench into the way we test on mobile platforms that you may not think about when transitioning from web app and UI testing to Mobile Testing.

The goal as a mobile applications tester is to ensure that the software that you and your team are releasing provides value to the end user. Also, the tester wants to minimize or eliminate bugs, crashes, poor performance, poor speed, battery drains, "slow" behavior or long startup times, and complex or confusing UI and UX.

In this paper we will discuss the following:

- Web App Testing vs Mobile App Testing
- Mobile operating systems
- Differences between native apps, hybrid apps and web apps
- App store submissions
- Testing considerations
- Frequency of use and app value

- Emulation vs Physical devices
- Hardware testing including
    - Geofencing and Location testing
    - push notifications
    - data connections
    - device level interrupts
    - multiple apps running
- Beginning Mobile Test Automation

3 Web App Testing vs Mobile App Testing

A Mobile App Tester needs to keep a frame of reference of the user, the context and intent of the user, such as

- When they are using their device
- Where the user is while using their device
- How much time they have
- How focused can they be

The time that a user spends in front of their mobile device can be greater than the amount of time they spend in front of their desktop computers as an aggregate. However, a user typically uses their mobile device in short spurts to accomplish very specific tasks. This would include acquiring relevant information in the shortest amount of time that is relevant to the task they have at hand.

An example of this would be a user checking the weather for the day when they first wake up to see if they need an umbrella, a jacket, or to put on shorts.

Another example would be a user making a reservation at a restaurant with a Restaurant Reservation Application, such as Open Table, sending a message to their friends about the reservation, with WhatsApp, and navigating to the restaurants with a real time update of their location that they share as they drive to the restaurant to meet their friends, with Waze.

Mobile Devices have many built-in inputs, sensors, and outputs. This enables some very creative uses of the mobile devices, but also adds complexity and variables to testing.

3.1.1 Web App Testing vs. Mobile App Testing

	Web App Testing	**Mobile App Testing**	
Categories		**Android**	**iOS**
OS	Windows, Mac, Linux, Unix, others	Custom overlays on top of AOSP Samsung, LG, Sony, Motorola etc.	iOS
Users Updates to OS		Varies by Manufacturer and Carrier	typically 3 months to latest version
Device	Typically x86, x64 processors	Varies by Manufacturer	Apple iPhone, iPad, iPad Mini, iPod Touch
Inputs	Keyboard, Mouse, webcam, microphone, fingerprint scanner	touchscreen, soft buttons – (home, back, menu), soft keyboard, front facing camera, back facing camera	touchscreen, soft buttons, soft keyboard, front facing camera, back facing camera
Outputs	screen, speaker	screen, speakerphone, headphone jack, haptic feedback	screen, speakerphone, headphone jack, haptic feedback
Buttons		power button, volume up/down	power button, volume up/down, home button
Sensors		light sensor, Accelerometer, Gyroscope, Pedometer, Compass, Hall, Fingerprint ID, Gesture, Barometer, Step detector, Step counter	Three-axis gyro, Accelerometer, Proximity sensor, Ambient light sensor, Fingerprint identity sensor, Home/Touch ID sensor, Backside illumination sensor
Location	from network/ip address	GPS, A-GPS, Glonass	Assisted GPS and GLONASS, Digital compass, Wi-Fi, Cellular
Connectivity	Wi-Fi or LAN, Bluetooth	Wi-Fi, Cellular, Bluetooth	Wi-Fi, Cellular, Bluetooth
Network Consistency	consistent	can be intermittent	can be intermittent
Screen Size	varies	varies	varies
Screen Resolution	Varies	Varies	Varies
Battery		varies 10 to 20 hours	varies 10 to 20 hours

Table 1.0 - Web App Testing vs Mobile App Testing Comparison Chart

In the Web App Testing vs Mobile App Comparison Chart (Table 1.0), mobile devices have sensors such as a GPS, Accelerometer, Barometers and light sensors that a traditional laptop or PC might not include natively. For Android, Google categorizes sensors into three Types of Sensors according to Google's Android Developer overview "Sensors Overview" there are the following types of sensors:

3.1.2 Motion sensors

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

3.1.3 Environmental sensors

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

3.1.4 Position sensors

> These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

Keeping the types of sensors in mind helps to separate out the testing the tester will need to run, and may influence the decisions on testing with physical devices and/or simulators and emulators that will be discussed later in this paper.

4 Mobile Operating Systems

Android and iOS are the predominant operating systems with Android controlling 51.7% and Apple controlling 41.6% of the market share for a total of 93.3% of the US Smartphone subscribers according to "comScore Reports January 2014 US Smartphone Subscriber Market Share".

Each year Apple (at WWDC) and Google (at Google I/O) have developer conferences where they announce and release changes to the Operating System including adding new API's, libraries, and/or changes to the SDK. Android and iOS are updated to the general marketplace within a few months of the developer conferences every year.

The changes may be slight updates and tweaks or they may be major updates to everything from graphics, layout, and design concepts to adding access to additional sensors or hardware capabilities.

Major UI changes and requirements to the operating system can result in a major overhaul needed for the development team and their application in order to conform and follow new design guidelines and styling rules. Also, this might mean that API's that the application is using have changed or become deprecated. Other effects are, existing paradigms of the OS such as navigation within the application may change as well.

For Mobile Testing, one thing is constant, CHANGE! The operating system and different API's will change and as a mobile application tester knows, they will need to be able to adjust and adapt to new UI designs, different API calls, and possible changes in workflow.

4.1.1 Personal Experience – Transitioning from iOS 6 to 7 and Gingerbread to Jelly Bean

I have worked through testing of mobile applications from the transition of iOS 5 to iOS 6, and iOS 6 to 7. With Android, there were some changes to the UI and API's that developers were accessing transitioning from Android Gingerbread to Ice Cream Sandwich; and from Ice Cream Sandwich to Jelly Bean. After each Operating System Update the application that I was working on had to be updated and retested for the older version of the operating system as well as the latest and greatest version of the updated operating system.

Being cognizant of changes in the UI, UX, how the native applications operate, user expectations and norms is important to testing applications that adhere to design guidelines that make mobile applications intuitive for users.

A software quality tester is the advocate for the applications' end user. The tester needs to understand what the end user's expectations of

- How the mobile application should function
- How they will use the application
- What they are trying to accomplish with the mobile application
- Assist the user solving their problems or achieving goals

5 Native Apps, Hybrid Apps, Web Apps and Responsive Design

Developers and testers need to take into account if the applications that they will be testing are native apps, hybrid apps or web views of a responsive designed web site. The article “Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options” has describes the differences between Native apps, hybrid apps and web apps as follows:

Native apps are apps that are installed on the device, typically installed through an app store such as Google Play Store or Apple’s App Store; can access all of the devices inputs, sensors and outputs, and has full access to the OS’s API’s. Native apps perform the best and typically confirm to the commonalities of that operating system.

Web Apps are apps that use standard web technologies – typically JavaScript and HTML5. Developers can develop sophisticated apps with HTML5 and JavaScript. However, there are some limitations as the web apps cannot access some of the native device functionality although that is beginning to change as more mobile centric protocols are added to the web stack.

Hybrid apps are apps that embed HTML5 apps inside a thin native container.

	Native	**HTML5**	**Hybrid**
App Features			
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG
Performance	Fast	Slow	Slow
Native look and feel	Native	Emulated	Emulated
Distribution	Appstore	Web	Appstore
Device Access			
Camera	Yes	No	Yes
Notifications	Yes	No	Yes
Contacts, calendar	Yes	No	Yes
Offline storage	Secure file storage	Shared SQL	Secure file system, shared SQL
Geolocation	Yes	Yes	Yes
Gestures			
Swipe	Yes	Yes	Yes
Pinch, spread	Yes	No	Yes
Connectivity	Online and offline	Mostly online	Online and offline
Development skills	ObjectiveC, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript

https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

The rest of this paper will focus primarily on testing native mobile applications, and some of the key concepts will be applicable to the mobile software quality tester even if they are testing web apps and hybrid apps.

6 App Store Submissions

The majority of applications are distributed on the application stores:

App Store for iOS

Play Store for Android

There are several smaller app stores such as Amazon App Store, Samsung App Store or app stores / curated apps that are ‘approved’ by the cellular networks such as Verizon’s Android App Store, or AT&T’s Android App Store.

Application Submission

For apps on the Apple App Store, they must be submitted to apple and it can take up to two weeks for an app to be processed and approved before they will appear on the app store. If the developer is releasing an update to the application, the app needs to be re-submitted each time there is an update and uploaded to apple again. Once an application is approved, the developer can hold off on releasing it to the public or set it to publish immediately once it is approved by Apple.

Apps submitted to the Google Play store do not require any wait times. However, Google runs through a battery of automated tests to verify no malicious code or inappropriate API’s or permissions are used. Once it is approved, the app can be live on the Google Play Store, or a date can be set for “release” in the future.

Some considerations about testing apps after they have been accepted by the app store:

- Verify that the application can be downloaded from the app store
- Verify that the application be upgraded from a previous version of the app store
- Verify that the application that is on the app store allows registering a new user
- Verify that the application allows a user to login

6.1.1 Others’ Experience – Checking Production Builds on the App Store

It’s good to verify the above in production to ensure that there were no changes to the code when switching from qa / development branches to putting the application into production. There have been issues with people pointing to a qa or dev endpoint rather than a production end point when publishing their application. Data is added to a development database rather than saved to the production database. Another possibility is a last minute change creeps in after QA has completed testing and the application crashes when trying to register.

Developing deployment processes, testing and verifying that everything goes to production (in the case of a mobile app, the app store) as expected is critical because otherwise all of the testing and development work will not reach the end user and will not provide value to the business.

7 Testing Considerations

7.1 Android

Android allows a lot of customization and flexibility. That is one of the beauties of having an operating system that is open source, although a large majority of the code is developed and led by Google. However, the openness also makes it much more challenging for developers and software quality assurance testers.

7.1.1.1 Android Overlays

Each manufacturer has a different overlay to the standard open version of Android Open Source Project or AOSP that they may add additional API calls to their specific hardware or implement a couple of things differently.

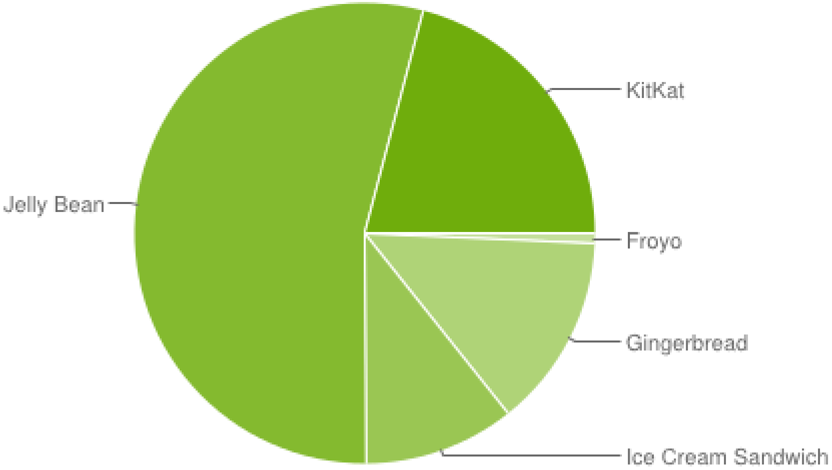
Different types of UI's compared, see PhoneArena.com's post "HTC Sense 6 vs Samsung TouchWiz vs Sony Xperia: UI comparison", to see the graphical differences between HTC, Samsung and Sony's Android Overlay.

Manufacturer	**Launcher/Overlay**
Samsung	TouchWiz
HTC	Sense
Sony	Xperia
LG	Optimus UI

Despite all the talk about how fragmented Android is, it is becoming less of an issue because:

1) Much of the standard API's are being incorporated into the Google Play Services which every device that is running the Google Play store utilizes. See Ron Amadeo, "Balky carriers and slow OEMs step aside: Google is defragging Android", in Ars Technica for more details

2) The Android Dashboard shows that devices using 4.1.x Jelly Bean and above totals 75.1% as of August 20, 2014 (According to the Google Dashboard which is updated every 7 days). If Ice Cream Sandwich version 4.03-4.04 is included in that number, over 85.7% of the android users. There have been some minor changes between Ice Cream Sandwich to Jelly Bean and Kit Kat.



http://developer.android.com/about/dashboards/index.html

For Android there are still a small portion of users 13.6% that are still using Gingerbread, but the majority of users are using Ice Cream Sandwich or above. To help the developer determine what level of API level to utilize, they may want to consider:

Does the application need the service calls that are made available in the latest API level?

Some considerations with testing mobile apps:

- What level of API is the app targeted towards
- Should tablets be considered as a target platform

Physical Device Considerations:

- older devices that have slower processors, or lower specifications including RAM and on-board lesser amounts of memory
- smaller sized screens – this will allow you to test your layout and UI
- larger sized screens – with the advent and popularity of phablets, the tester may want to make sure that the layout can scale, doesn't look awkward on larger sized screens including tablets
- Different manufacturers and their various UI overlays
- Test older operating systems (SDK's on the Simulators) on older devices
- Test the latest operating system with the latest devices
- Physical Keyboards
- Other Built in Accessories specific to the device hardware

Testing devices with physical keyboards, built in accessories like the S-Pen for Samsung note device, or infrared blasters. If your application is highly dependent on text input, it is a good idea to include testing with devices that have physical keyboards. Some enterprise companies have switched from using BlackBerry devices with physical keyboards to android devices with physical keyboards.

7.1.1.2 Personal Experience - Android Operating System Testing Jelly Bean vs Kit Kat

For the mobile app that I work on, the current minimum target API level is Gingerbread (Android 2.3.3 or API level 10); testing with a Kit Kat device our team was able to locate a defect with how data was handled in an array from a JSON response from the Web Server. The development team had used a Nexus device that was running Android Jelly Bean and did not see any issues. However, testing with a Nexus 5, running Android Kit Kat, the response returned was different with Android Kit Kat. The Android developer was able to iterate through the array of returned data from the web service call and match the relevant response rather than relying on the call to return the items in a certain order which worked fine on the developer's Android Jelly Bean device.

7.1.1.3 Latest Devices

If it is available, and within the company budget, test on the most popular devices for the year, i.e. best sellers and flagship devices because these will be the devices that the majority of the audience will be utilizing the app with most frequently. Most of the time these devices will not have issues with the processing power or memory; the issues will typically be around intricacies in how the overlays, and customized launchers handle the interactions of the application and some device specific hardware that may not be common to all devices. Some examples of newer sensors:

- Heart rate monitor
- Fingerprint ID
- Environmental Sensors
  - Thermometer
  - Relative Humidity
  - Pressure Sensor

7.1.1.4 Lower End Devices

It is a good idea to obtain a few older devices for compatibility testing. Testers can acquire devices on the lower end of the spectrum with the minimum API level that the application supports.

Make sure that the devices have a variety of:

- screen sizes including smaller screens
- lower RAM specifications
- slower processors
- physical keyboards

7.1.1.5 Personal Experience – Variety of Devices

Early on in testing our application, we obtained a Pantech Pocket. It looks almost square shaped, runs Android Gingerbread 2.3.4, has a resolution of 600 x 800 pixels, has a 1 GHz Qualcomm Snapdragon S2 processor and 512 MB of RAM. It was a good test device, because it had lower specifications, had an oddly shaped screen, and made by a smaller manufacturer.

7.1.1.6 Alternatives to Building a Device Lab

If the tester's company does not have ability to purchase, rent, or lease some of the latest devices, there are testing services that allow the tester to push their application to a vendors' set of test devices such as AppThwack https://appthwack.com/ or Test Droid http://testdroid.com/ and run tests on their set of hardware.

There are also services that allow running tests on a battery of pre-configured emulated devices such as:

Sauce Labs - https://saucelabs.com/mobile

Perfecto Mobile - http://www.perfectomobile.com/

SOASTA - http://www.soasta.com/

7.2 iOS

The variability for iPhones is a little lower compared to Android. However, there are some things that the tester should keep in mind as they begin testing iPhone, iPad and iOS Apps.

Test with larger and smaller screen sizes. This includes testing with iPad's and iPad Mini's to see how your application scales with the 2x button.

7.2.1 Personal Experience – iPhone Screen Size

In testing our application, there were some defects where certain parts of the screen looked fine for the iPhone 5, and 5s; but the keyboard covered a field, or a section at the bottom of the screen that was not visible because of the smaller screen size for the iPhone 4 and 4s. Another example, on certain screens of the application there were a couple of places where the user would need to scroll vertically on an iPhone 4s in order to view the content at the bottom, however the view was not scrollable. Everything was visible testing with an iPhone 5 and 5s.

If your application supports a minimum level of the OS, test with older versions of the Operating System as well as the latest version to ensure that there weren't any defects related to updates or changes in the OS.

7.2.2 Personal Experience – iOS 7 Updates

In one of the updates to a version of iOS 7, there was a defect from Apple that had to do with the list views that caused a line with pricing information not to be displayed. We found the defect and added a patch to our mobile app. A couple months later Apple updated their defect, and their update broke our patch, so we had to update our code to take out the fix that was added. Staying on top of updates and patches is important for testers.

iPhone users update their phones when a new release is distributed fairly quickly because it comes directly from Apple. According to Kukil Bora, "Apple iOS 7 Two Months After Release: Adoption Rates Cross 70% Mark, Surpassing Those Of Predecessor's" in IB Times and Chitika's "UPDATE: iOS Version Distribution Study - 2012 through 2013, 3 months after iOS 7 was released 70% of users updated their devices to the latest OS. Some older devices did not meet the minimum requires specifications, and were not able to be updated.

8 Application Value and Frequency of Use

Mobile App Testers want to ensure that the user has a good user experience, and will come back to the application the next time they need to interact with or solve the problems that the application assists them in resolving. If the user has a bad experience, they will uninstall, delete and even give the app a poor rating. Continuing to provide value to the user is critical to a good user experience.

- What is the frequency of use?
  - Used Frequently
  - Used Occasionally
- What is the value of the application?
  - Acquire relevant information
  - Provide access to important content
  - Assists in tracking or monitoring some personal data?
- Does it simplify the user's life?
  - Reminders
  - Directions

Some apps are used on a daily or multiple times a day such as social media, news, weather or other timely information apps

Other apps are used a couple of times a month such as banking apps to check on the status of your account or make sure that a payment has been processed.

The apps that are used infrequently need to provide a value to users. Most of the time an app only stays on the smartphone as long as it is providing value. If it stops providing value or crashes the user will delete it. Which apps do testers have that were installed, used once or twice to explore and then immediately deleted, because they were buggy, or did not fulfill the desired need?

If a tester takes a look at the apps on their phone:

- which ones do they use frequently
- Which ones do they use occasionally
- Which ones only a couple of times a year

The tester wants to ensure that the app is valuable to the end user whether it is used frequently or occasionally. The app must provide some value to the user such as relevant content, entertainment, ability to communicate or share.

- Is your app used a couple times a year, but still has value because it simplifies the life of the user?

8.1.1 Personal Experience – Simplifying Life of Users and High Value

The application that I work on is estimated to be used about 3.4 times a year for the average user. It has a high value because it simplifies the users' life, and assists them in keeping track of their personal health information. The app allows users to easily interact with their personal data, have better conversations with their healthcare providers and can act as a reminder to ask additional questions on the next visit. The app provides enough value that they do not delete the application even though it is not used on a daily basis.

- Is the UI intuitive enough for the user to come back to, if they use the app infrequently?
- Does it follow the design principles, UI and UX prescribed by Apple or Google?

Mobile app testers want to ensure that the app is easy for users to figure out how to use without complex instructions or tutorials. The application should conform to common design practices, be intuitive, and simple to understand even if your apps are not simple underneath the covers.

9 Emulation vs Physical Device

There is a lot of discussion about using real physical devices and emulation. Physical devices are what the consumers will be using and as a tester, you should try to simulate the end user experience as much as possible. However, the devices are designed to be consumer grade level devices, and are not necessarily designed to be run 24 hours a day, 7 days a week as a test device. The devices will eventually fail and you may discover flakiness due to hardware issues, radio issues, heat etc. especially if the device is older, or has been used heavily for testing. Therefore, it is important to have a good mixture of emulation / simulators and physical device testing.

In Thomas Knych's portion of the GTAC 2013: Breaking the Matrix – Android Testing At Scale presentation, he discusses testing automation using physical devices vs emulating devices. See the section starting at https://www.youtube.com/watch?v=uHoB0KzQGRg#t=812

It's worth watching if the tester is unsure about the pros and cons of emulation/simulation vs physical devices. The basic conclusion is to try to do as much testing on a simulated device as possible, because the tester can spin up a variety of screen sizes, API levels, and memory configurations at a lower cost than on physical devices. Physical device testing should primarily be used for testing the areas that simulators cannot do, or don't do well.

9.1 Personal Experience – Simulator

The Android developer on our team did not have access to a device running Android Kit Kat, as he had a Nexus with Android Jelly Bean. One of the defects that QA found only occurred with a Nexus 5 running Android Kit Kat. The Android developer was able to reproduce the Kit Kat defect, debug and implement a fix using a simulator.

A physical device may be necessary if your application is explicitly using a sensor or input that cannot be easily simulated. Many applications use location services tied to GPS and Wi-Fi, camera, or microphones. Some of the latest devices include accelerometer, gyroscope, temperature, pressure, magnetic field, light, step counter and relative humidity and some of these are not be easily simulated. See information about inputs in "Sensor" article of Android Developer Hardware Reference.

If the app uses GPS, you can use mock locations as long as you use a simulator that has google play services enabled.

HAXM suggestion

If you are worried that simulators are slow, you can enable and use Intel's hardware image to speed up the simulator. It works for both Windows and Mac machines. It's called Intel Hardware Accelerated Execution Manager or HAXM for short.

https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager

for windows machines, you may need to boot into the bios and enable virtualization. However, once that's done you can be speeding along with a simulator.

In the GTAC 2013: Breaking the Matrix – Android Testing at Scale, they had a demonstration where tests were run on a HAXM enabled simulator that ran faster than a physical device (Nexus 4).
https://www.youtube.com/watch?v=uHoB0KzQGRg#t=1434

10 Hardware Testing

Some considerations for testing with hardware include location services such as GPS and Wi-Fi, push notifications, data connections and device level interrupts.

10.1 Geofencing and Location Testing

iOS seems to have pretty good location services and is fairly consistent with the location tests, due to aGPS and Wi-Fi being on by default for users. With Android, there is quite a bit of variance on how well the GPS operates from device to device. The Wi-Fi assistance and variability in the hardware / chips that are used based on a manufacturer's. Some android devices can take a couple of minutes to lock onto a location, but if you are indoors it may not lock onto a location with GPS even after 10 to 15 minutes.

10.1.1 Personal Experience – Locations Testing

For the application that I test, we ran some tests with GPS inside the office building, and I was not able to get any GPS lock even after 10 minutes. I tested placing multiple devices next to the window and walking around the office building, and still was not able to get a GPS lock. However, within one to two minutes of walking outside I was able to obtain a GPS lock. If your app uses location services, you and your developers need to be how accurate location services are and how long it will take and how quickly the *device will lock onto a position. Also, there are power and latency considerations as well. Both iOS and Android have coarse and fine ratings for their location services.*

10.2 Push Notifications

10.2.1 Push Notifications on Android Simulators

For Android, push notifications can be received on a simulator, but the simulator will need to have google play services enabled. In the Android SDK Manager they will say Google APIs (x86 System Image) or (ARM System Image).

For more information see the following on stackoverflow.com “Push notifications don't work”, and “Android emulator not receiving push notifications”.

10.2.2 Push Notifications on iOS Simulators

For iOS, push notifications cannot be sent to simulators, so the tester will need to use a physical device. An iPhone, iPod Touch or iPad on Wi-Fi will work. The device does not necessarily have to have a cellular connection and will still be able to receive push notifications.

For more information see the following on stackoverflow.com “can we check push notification in simulator? [duplicate]”, and on stackexchange.com ““Is it possible to send push notifications to the iPhone Simulator?”.

10.3 Data Connections

As a Mobile Tester, it is important to consider the intents of the user and most likely where they will be using your application. If your application relies on a data connection, you will need to take into account how the application behaves when

1. The cellular connection is intermittent
2. The user switches from cellular to Wi-Fi
3. The user switches from Wi-Fi to cellular
4. How other applications utilizing data could affect the data performance of your application

Network Consistency can be a consideration that new mobile app testers don’t think about too much. With a Web App, typically the user expects the website to have excellent uptime, and just work. With cellular connections and switching between cellular and Wi-Fi, the network connectivity needs to be taken into account for your testing.

10.4 Device Level Interrupts

Mobile testers need to take into consideration device level interrupts such as:

1. Phone calls coming in and how the app handles going into the background
    a. Resuming after the user has completed the telephone call
2. Text messages coming in and how the app handles going into the background
    a. resuming after the user has completed reading and replying to a text message
3. low battery
    a. does the app decrease the frequency it tries to access the data network if the battery status is low
    b. does the app shut down or go into the background
4. notifications

    a. how does the application handle notifications that occur in the background and the user accesses the notifications

10.5 Multiple apps running

How does the app handle multiple applications being open?

Does the app gracefully handle going into the background?

If there are many applications opened up and the memory a majority of the available memory is used up, does the app minimize the amount of memory?

If the user runs some of the commonly used system level and third party applications such as maps, calendar, weather, news, does the app perform properly?

If the app interacts with other apps, are they handled properly such as:

1. sharing links or articles
2. launching a web browser
3. launching the email application with some of the information pre-configured for to address and subject line
4. opening phone numbers in the dialer
5. launching a map application to navigate to a particular location
6. voice search using microphone input
7. image capture using camera or front facing camera
8. capturing video

If there is music or sounds effects for your application, how does it interact with system level sounds?

How does the app interact with applications that use sound such as music, maps, voice search?

11 Beginning Mobile Test Automation

As a Mobile App Tester, trying to determine what to manually test and what to automate is a balancing act. When you embark on your mobile app testing journey, you will most likely have to do a lot manual exploratory testing. As you progress and the testing process matures you will gain an understanding of the core functionality of the application under test.

The mobile tester can start automating the basics and build up a set of tests that can cover all the areas that would qualify for a smoke test to provide feedback to developers in a quick and timely manner.

Before the Mobile App Tester begins to automate any tests, it's

1. take a look at some of the options for mobile test automation
2. determine the requirements and desired outcomes of automated mobile testing
3. compare the pros and cons of the various tools to the needs and desires of the development team, product owner, and business value
4. Assess the familiarity of the test writers to programming and the various programming languages that may be involved with writing test automation.
    a. Will QA people with experience be writing and maintaining the tests?
    b. Will developers be writing some of the automated tests?
    c. Will any non-programmers or non-technical people need to look at tests or interpret test results?
5. Decide on a test framework and test strategy

6. Try a proof of concept for the selected testing framework
7. Evaluate or re-evaluate if the framework will fulfill the requirements
    a. If it fulfills the requirements, continue to write additional tests to cover a smoke test of the mobile app
    b. If it does not fulfill the requirements, find the next testing framework down the line and repeat steps 5-7 until the tester finds a testing framework that fulfills the requirements
8. Periodically re-visit the testing frameworks, and updates to see if the tools have improved.

One of the simplest tests to start writing is logging into your application. The next one may be registering a new user. After that, look to automate some of the most common things that a user would need to perform within the app; this can include CRUD (Create Read Update Delete functionality) for various aspects of the application. After that has been completed, start adding additional features and hooks to build out the testing framework specifically for the mobile application.

Currently, there is no one size fits all solution for automated mobile app testing. Each team seems to come up with their own solutions that seem to work best for their situation.

Some of the Testing frameworks that

Android

- Robotium
- Espresso
- UiAutomation

iOS

- UI Automator
- KIF
- Frank
- iOS Driver

Both Android and iOS

- Appium - http://appium.io/
- Calabash - http://calaba.sh/
- Monkey Talk - https://www.cloudmonkeymobile.com/monkeytalk

11.1.1 Personal Experience – Beginning Mobile Android UI Automation

For our Android mobile automation QA initially investigated Appium. We tried setting up Appium on Windows and ran into difficulties starting and running Appium on windows. Then, the development team tried Robotium and was able to get it started in a day or two. We built up a testing framework with tests that covered the typical smoke test and core areas of our mobile app. However, there were some issues with consistencies and intermittent failures.

We attempted to incorporate Jenkins and CI on a virtual machine and ran into major issues with timeouts and test failures. The tests would run without issues locally, but on the VM the tests would time out even after increasing global time outs significantly. After GTAC 2013, there was information about Android Espresso that was written by Google employee, Valera Zakharovand. The development team tried a proof of concept and transitioned to using the Espresso Framework a couple of months after the release of Android Espresso. QA and the development team have been adding tests to the automation suite and working on incorporating CI with Jenkins.

You can find out a little more about Espresso:

https://code.google.com/p/android-test-kit/

In writing tests and helpers for mobile test automation, some different styles of the framework may include:

- page object model, or in the case of the app a screen object model
- flow or path dependent based on activities of the app

Some advantages of the screen object model, is that all elements on a screen will be accessible for the screen and can be easily reused. One disadvantage is that some of the helper methods can have very little functionality, and it can get complicated to navigate between different menus or screens if the tester needs to chain together a sequence of multiple screens.

The Flow or path methodology works well if there are many different steps in the application that is being tested and there are many screens that would need to be accessed. It is a good idea to try to encapsulate methods that can be re-used into helper files. One of the disadvantages of flow testing is that if there is a slight variance in the test a new flow or path of the test will need to be initiated.

11.1.2 Personal Experience – Screen Object Model and Flow Testing

When the QA and development team started with the Robotium framework, we borrowed a lot from our experience with Selenium and decided to go with a Page Object Model / Screen Object Model. It seemed to work well during the first couple of iterations of the mobile application. However, as more functionality of the app was added it seemed to get heavier and a bit cumbersome.

When the development team and QA decided to transition to Espresso, we decided to attempt to write tests in a flow methodology with each test containing the full flow of the test. IN the beginning, there wasn't as much encapsulation of functionality as we would have liked.

Eventually the development team and QA concluded on doing a combination of Screen Object Model and Flow testing for the mobile application depending on the complexity of the particular test. Often times, there would need to be 4 or 5 things that would need to be setup before the functionality that was under test could actually be tested. The common setup steps ended up being a part of the flow testing and the new functionality would have more of a screen object model paradigm applied to it.

Mobile App Test Automation is a changing process and new tools and better methodologies continue to appear as the mobile app test automation space continues to evolve and mature.

12 Conclusion

Mobile App Testers can see that there are a variety of considerations to take into account when creating and implementing the mobile testing strategy. If the mobile app tester is coming from an n-tier application architecture or web app testing environment, the web services and back end still function the same. The main difference is the presentation tier is on the smartphone rather than on a webpage of a desktop or laptop computer. Many of the same skills and techniques can be re-purposed for mobile app, and web app testing on mobile devices.

This paper discussed WebApp Testing vs Mobile App Testing; the dominant Mobile Operating Systems of iOS and Android; differences between native apps, hybrid apps, and web apps; app store submissions; mobile app testing considerations; frequency of use and app value; Emulation vs Physical devices; Hardware testing; and Beginning Mobile Test Automation.

After reviewing this paper the mobile tester should have a better idea of all of the considerations and things that they might not initially had before they sit down and begin testing. They can review the questions about testing strategy, devices, emulation vs physical devices and map out a testing strategy that will fit into the constraints of the business, budget, and time availability. Test and try out many different apps and become comfortable with exploratory and manual testing.

Mobile app testing, can be a challenging arena due to the many changes of hardware, operating system, UI changes, UX changes, market expectations, and the constant upgrade cycle of users. It is also rewarding to see the application being used and providing value for users and knowing that the mobile tester contributed to an intuitive and enjoyable experience for the end user.

This is definitely an exciting area to be doing software quality assurance and testing, at the same time the tools and platform that the mobile tester is using today and will use tomorrow is constantly changing. Many of the problems and tools that are still being developed today will help solve the issues that mobile testers are encountering today and in the future. Just as web app testing took time to mature over the last several decades, mobile testing will evolve and the tools and best methodologies will be firmed up. Testers will see tools mature and industry accepted tools similar to what we have with web service and web app testing.

13 References

“comScore Reports January 2014 US Smartphone Subscriber Market Share”, comScore.com, last modified March 7, 2014, accessed June 16, 2014, https://www.comscore.com/Insights/Press-Releases/2014/3/comScore-Reports-January-2014-US-Smartphone-Subscriber-Market-Share

“One In Every 5 People In The World Own A Smartphone, One In Every 17 Own A Tablet [CHART]” Smartphone And Tablet Penetration - Business Insider”, BusinessInsider.com, last modified December 15, 2013, accessed June 16, 2014, http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

“Smartphone Users Worldwide Will Total 1.75 Billion in 2014”, eMarketer.com, last modified January 16, 2014, accessed August 18, 2014, http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536

Mary Meeker, "Internet Trends 2014 - Code Conference" (slides presented at 2014 Internet Trends by Kleiner Perkins Caufield Byers), http://www.kpcb.com/internet-trends

"Sensors Overview" Android Developer Sensors Topics, accessed June 16, 2014, https://developer.android.com/guide/topics/sensors/sensors_overview.html

“Mobile: Native Apps, Web Apps, and Hybrid Apps”, Raluca Budiu, modified September 14, 2013, accessed June 16, 2014, http://www.nngroup.com/articles/mobile-native-apps/

“Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options”, salesforce.com Developer, accessed June 16, 2014, https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Chris P., “HTC Sense 6 vs Samsung TouchWiz vs Sony Xperia: UI comparison”, PhoneArena.com, modified March 25, 2014, accessed June 16, 2014, http://www.phonearena.com/news/HTC-Sense-6-vs-Samsung-TouchWiz-vs-Sony-Xperia-UI-comparison_id54269

Ron Amadeo, “Balky carriers and slow OEMs step aside: Google is defragging Android”, modified Sept 2 2013, accessed June 16, 2014, http://arstechnica.com/gadgets/2013/09/balky-carriers-and-slow-oems-step-aside-google-is-defragging-android/

Kukil Bora, “Apple iOS 7 Two Months After Release: Adoption Rates Cross 70% Mark, Surpassing Those Of Predecessor's”, modified December 4, 2013, accessed June 16, 2014, http://www.ibtimes.com/apple-ios-7-two-months-after-release-adoption-rates-cross-70-mark-surpassing-those-predecessors

"UPDATE: iOS Version Distribution Study - 2012 through 2013", Chitika.com, modified December 12, 2013, accessed June 16, 2014, http://chitika.com/insights/2013/ios-distribution-update

Thomas Knych, Stefan Ramsauer, & Valera Zakharov, "GTAC 2013: Breaking the Matrix – Android Testing At Scale", Google Test Automation Conference, modified April 29, 2013, accessed June 16, 2014, https://www.youtube.com/watch?v=uHoB0KzQGRg

"Sensor", Android Developer Hardware Reference, accessed June 16, 2014, http://developer.android.com/reference/android/hardware/Sensor.html

Christopher Lawless, "Android emulator not receiving push notifications", stackoverflow.com, modified December 11, 2013, accessed June 16, 2014, http://stackoverflow.com/questions/20521600/android-emulator-not-receiving-push-notifications

user1242617, "Push notifications don't work", stackoverflow.com, modified May 23, 2014, accessed June 16, 2014, http://stackoverflow.com/questions/16713321/push-notifications-dont-work

Rahul Vyas, "can we check push notification in simulator? [duplicate]", stackoverflow.com, modified November 28, 2009, accessed June 16, 2014, http://stackoverflow.com/questions/1811900/can-we-check-push-notification-in-simulator

Simon, "Is it possible to send push notifications to the iPhone Simulator?", stackexchange.com, modified March 5, 2012, accessed June 16, 2014, http://apple.stackexchange.com/questions/42654/is-it-possible-to-send-push-notifications-to-the-iphone-simulator