Mastering Mobile Test Automation

Mobile testing and the cross-platform validation of applications is both a challenging and lucrative area of expertise where organizations aspire to leverage automation. The book begins by teaching you the basics of mobile testing and automation to help you develop an in-depth understanding of mobile automation concepts and techniques. Moving on, the scalability is put into focus by taking you through learning mobile automation techniques with web browsers and progressing on to emulators and simulators. Helping you get accustomed to physical as well as cloud automation, the book lays a solid foundation for advanced topics that involve optimization and strategizing to deliver state-of-the-art solutions.

Leading test automation tools such as UFT (formerly QTP) and Selenium - the leader in open source tools, along with specialized mobile test automation tools such as ExpertTest SeeTest and Cloud-based mobile automation solutions like Mobile Labs and Perfecto Mobile have been used to explain the concepts. By the end of the book, you will have an in-depth understanding of mobile automation techniques as well as the solutions to various challenges faced.

What you will learn from this book

- Explore and configure the various types of mobile application packages for mobile test automation
- Automate mobile web applications through user agent simulation technique
- Analyze and interpret different types of test automation frameworks with respect to the mobile platform automation
- Implement techniques and set up labs to deliver a sustainable and comprehensive mobile test automation environment
- Automate and test using physical devices as well as emulation and simulation of mobile platforms
- Get in depth understanding of cloud based mobile testing and automation techniques
- Configure, manage, and maintain a mobile automation test suite for optimum productivity and Return on Investment
- Troubleshoot and employ industry best practices in a mobile testing environment

Who this book is written for

If you want a complete understanding of mobile automation testing and its practical implementation, then this book is for you. Familiarity with the basics of VB Script and Java along with knowledge of basic testing concepts is essential.

In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Ensuring Five-star Rating in the Marketplace'
- A synopsis of the book’s content
- More information on Mastering Mobile Test Automation

About the Authors

**Feroz Pearl Louis** is an automation architect with over 10 years of extensive experience working with automation tools, such as HP UFT (formerly QTP), Selenium, SOAtest, SoapUI Pro, Cucumber, Watir, Ranorex, SeeTest, Perfecto Mobile, and Mobile Labs. His core expertise is in functional and mobile automation with traditional and nontraditional tools and techniques. He is currently working with the Automation Centre of Excellence of a Fortune 500 IT services company. He leads a team of 40 plus people spread across the entire spectrum of automation services, including functional, SOA, mobile, cross-browser and cross-platform automation, and Agile automation with behavior-driven development methodology. With his extensive design and development experience, he engages with various customer teams to design, implement, and deliver solutions that address their specific pain points. He is a prolific contributor to the automation community on various forums, such as LinkedIn, Quora and SQA Forums, and blogs at [https://ferozlouis.blogspot.com](https://ferozlouis.blogspot.com).

**Gaurav Gupta** is a dynamic and young IT professional who is engaged in the testing of web and mobile applications. He has worked with various tools and techniques in the broad field of mobile testing. He has extensive experience in both web and mobile test suite design and optimizes various operating systems. He is a versatile tester and is always keen to learn new technologies to keep himself updated. His passion for work makes him stand out among others. Even at a relatively early stage of his career, he is the published author of the book *Mastering HTML5 Forms*. A graduate in Computer Science, he works with a reputed Fortune 500 IT services company. Gaurav is a native of Chandigarh, India, and currently lives in Pune, India.
Mastering Mobile Test Automation

This book will guide you to develop an in-depth understanding of mobile automation concepts and techniques to design, implement, and deliver customized mobile automation solutions with real, emulated devices and cloud-based mobile testing tools, such as Perfecto Mobile, and Mobile Labs, integrated with traditional tools, such as UFT and Selenium, and other leading tools in the market, such as ExperiTest SeeTest.

This book is a comprehensive reference guide to every aspect of mobile test automation and can be used in any organization that is engaged in test automation and/or takes up mobile test automation.

What This Book Covers

Chapter 1, Ensuring Five-star Rating in the MarketPlace, explains how mobile test automation is different from traditional functional automation and what the typical challenges are that you have to overcome. This chapter provides you with a brief overview of the different techniques used for mobile automation. These techniques are elaborated in later sections of the chapter. This section explains why and how there are different techniques.

Chapter 2, Designing Mobile Automation Frameworks, explains the various types of automation frameworks with respect to mobile automation, such as functional decomposition, keyword-driven, data-driven, and hybrid, with a focus on mobile-specific differences. This has real examples of working code for various tools.

Chapter 3, User Agent – automating Mobile Applications with Browsers, provides you with the opportunity to learn the use of the user-agent technique, along with the code snippets and lab setup for this technique. We will also delve into the advantages and disadvantages of this technique.

Chapter 4, Emulators and Simulators – the Automation of Emulated Devices, explains the use of emulators and simulators for mobile test automation, along with the code snippets and lab setup for this technique. We will also delve into the advantages and disadvantages of this technique.

Chapter 5, Automating Physical Devices, showcases the techniques required to automate a mobile application with physically present real mobile devices.

Chapter 6, Automating on Cloud, provides you with the opportunity to learn the techniques required to automate a mobile application with cloud tools such as Perfecto Mobile and Mobile Labs.
Chapter 7, *Optimizing Test Strategy and Estimation*, deals with the selection of the most suitable strategy to focus on maximizing the return on investment for automation and the ways to avoid any potential pitfalls. We will also look at the various technical aspects that are usually overlooked and only become apparent at the end of a delivery cycle. This chapter deals with how to estimate your mobile automation effort.

Chapter 8, *Delivering Customer Delight*, sums up all the lessons learnt in the previous chapters and covers the test strategy planning, project effort optimization, ways to maintain the automation suite, resource identification, and the management of hardware and software.
Ensuring Five-star Rating in the Marketplace

The star rating system on mobile marketplaces, such as Google Play and Application Store, is a source of positive as well as negative feedback for the applications deployed by any organization. This system is used to measure various aspects of the application, such as like functionality, usability, and is a way to quantify the all-elusive measurement-defying factor that organizations yearn to measure called "user experience", besides the obvious ones, such as the appeal and aesthetics of an application's graphical user interface (GUI). If an organization does not spend time in testing the functionality adequately, then it may suffer the consequences and lose the market share to competitors. The challenge to enable different channels such as web applications through mobile browsers, as well as providing different native or hybrid applications to service the customers as per their preferences, often leads to a situation where organizations have to develop both a web version and a hybrid version of the application.

At any given point of time, it is almost impossible to test an application completely, and to cover various permutations and combinations of operating systems, their versions, device manufacturers, device specifications with various screen sizes, and application types, with solely employed manual testing techniques.

This is where automation comes to the rescue. However, mobile automation in itself is very complex because of the previously explained fragmentation issue. In this chapter, you will learn how not to fall into the trap of using different tools, frameworks, and techniques to address these differences.
In this chapter, we will cover the following topics:

- Introduction to mobile test automation
- Types of mobile application packages
- Mobile test automation overview
- Some common factors to be considered during mobile testing, including interrupt testing, form factor testing, layout testing, and more
- Overview of different types of mobile automation testing approaches
- Selection of the best mobile testing approach depending on the project
- Troubleshooting and best practices

**Introduction to mobile test automation**

Before we start learning about mobile test automation, let's understand what functional test automation is.

Test automation has always been a fundamental part of the software testing lifecycle for any project. Organizations invariably look to automate the repetitive testing actions in order to utilize the manual effort thus saved for more dynamic and productive tasks. Use of automation tools also allows utilization of system idle time more effectively. To address these needs, there are a plethora of tools available in the market along with various frameworks and implementation techniques. There are both open source and licensed tools available in the market. Tools such as HP’s **Unified Functional Testing (UFT)**, formerly known as **QuickTest Professional (QTP)**, **TestComplete**, **Selenium**, **eggPlant**, **Ranorex**, **SilkTest**, **IBM Functional tester**, and numerous others, provide various capabilities for functional automation.

However, almost all of these tools are designed to support only a single operating system (predominantly Windows—owing to its popularity and the coverage it enjoys across industry verticals), although a few provide support for other lesser-used operating systems, such as Unix, Linux, Sun Solaris, and Apple Macintosh.

As far as functional automation is concerned, you don't need to even consider the implications of supporting multiple operating systems in most cases. With Windows as the only operating system that is supported, there aren't any considerations for different operating systems. If the application is a web application, then there may be a need to do cross-browser testing, that is, testing automation on various browser types (Chrome, Firefox, and Safari besides Internet Explorer) and their respective versions.
Also, as far as functional automation is considered, there is a very clear demarcation between nonfunctional and functional requirements. So, an automated solution for functional testing is not required to consider factors such as how other processes running on the machine would impact it, or any of the hardware aspects, such as the screen resolution of monitors and the make of the machines (IBM, Lenovo, and others).

When it comes to mobile automation, there is an impact on the test suite design due to various other aspects, such as operating systems (Android, iOS, Blackberry, Windows) on which the application is supposed to be accessed, the mode of access (Wi-Fi, 3G, LTE, and so on), the form factor of the devices (tablets, phones, phablets, and so on), and the behavior of the application in various orientation modes (portrait, landscape, and so on).

So, apart from normal automation challenges, a robust mobile automation suite should be able to address all these challenges in a reliable way. In the coming chapters, we will learn about how to create such frameworks, but before that, we need to have a thorough understanding of the challenges.

Fragmentation of the mobile ecosystem is an aspect that compounds this manifold problem. An application should be able to service different operating systems and their flavors provided by original equipment manufacturers (OEMs), such as Apple with iOS, Google's Android with Samsung, HTC, Xiaomi, and numerous others, Windows with Nokia and HTC, and even Blackberry and other lesser-used operating systems and devices. Add to this the complexity of dealing with various form factors, such as phones, tablets, phablets, and their various hybrids.
Ensuring Five-star Rating in the MarketPlace

The following figure is a visualization of the Android market fragmentation over various equipment manufacturers, form factors, and OS versions:

As we know, test automation is the use of software to automate and control the setting up of test preconditions, execution of tests, test control, and test reporting functions with minimum, or ideally zero, user intervention. Automating the testing for any mobile application is the best way to ensure quality, and to achieve the quick and precise results that are needed to accommodate fast development cycles.

Organizations look toward functional test automation primarily to reduce the total cost of ownership over a period of time, and to ensure the quality of the product or application being developed. These advantages are compounded many times for mobile test automation and hence it provides the same advantages, but to a much greater degree.
The following are the various advantages of mobile test automation for any project:

- **Improved testing efficiency**: The same scripts can be used to run uniform validations across different devices, operating systems, and application types (of the same application), thereby reducing the test execution effort considerably. This also means that the return on investment (RoI), which typically takes about 3-5 cycles of executing the conventional functional automation to achieve breakeven, is viable in most cases within the first release itself, as mobile testing is typically repeated on many devices. So, in this case, fragmentation acts as a positive factor if the automation is employed properly, whereas, with pure manual testing, it greatly increases the costs.

- **Consistent and repeatable testing process**: Human beings tend to get bored with repetitive tasks and this makes such a test prone to errors. Due to the effect of fragmentation in the mobile world, the same application functionality needs to be validated across various combinations of operating systems, application types, device manufacturers, network conditions, and many more. Hence, the use of automation, which is basically a program, ensures that the same scripts run without any modifications every time.

- **Improved regression testing coverage**: The use of automation scripts allows the regression tests to be iterated over multiple combinations of test data. Such data-driven scripts allow the same flow to be validated against different test data combinations. For example, if an application allows users to search for the nearest ATMs in a given area, basically, the same flow would need to be tested with various zip codes as inputs. Hence, the use of automated scripts would instantly allow the test coverage to be increased dramatically.

- **More tests can be run in less time**: Since automated scripts can be run in parallel over various devices, the same amount of testing can be compacted inside a much smaller time window in comparison to the manually executed functional testing. With the use of automation scripts that include device setups as preconditions, the execution window can be exponentially reduced, which otherwise would take a manual tester considerable time to complete.

- **24/7 operation**: Although any functional automation suite can lead to better resource utilization in terms of executing more number of scripts in lesser time, with respect to mobile automation, the resources are often expensive mobile devices. If functional testing is done manually, then more of the same devices need to be procured to allow manual testers to carry out tests, and especially, more so in the case of geographically distributed testing teams. Mobile automation scripts, on the other hand, can be triggered remotely and can run unattended, reducing the overall cost of ownership and allowing 24/7 utilization of devices and tools.
- **Human resources are free to perform advanced manual tests**: Having automation scripts perform repetitive regression testing tasks frees up the bandwidth of manual testing teams for exploratory tests that are expensive to automate and cumbersome to manage. Hence, the use of automation leads to a balanced approach, where testers can perform more meaningful work and thereby improve the quality of delivered applications. In mobiles, since regression is more repetitive on account of the fragmentation problem, the amount of effort saved is manifold, and hence, testers can generally focus on testing aspects such as user interface (UI) testing and user experience testing.

- **Simple reproduction of found defects**: Since automation scripts can be executed multiple times on demand and are usually accompanied with reports and screenshots, defect triangulation is easy and is just a matter of re-execution of automation scripts. With pure manual testing, a tester would have to spend effort on manually recreating the defect, capturing all the required details, and then reporting it for defect tracking. With mobile automation, the same flow can be triggered multiple times on a multitude of devices hence, the same defect can be replicated and isolated if it occurs only on a specific set of devices.

- **Accurate and realistic real-life mobile scenarios**: Since a mobile requires tests to be specifically designed for variable network conditions and other considerations, such as device screen sizes, orientation, and more, which are difficult to recreate accurately with pure manual testing effort, automation scripts can be developed that accurately recreate these real-world scenarios in a reliable way. These types of tests are mainly not required to be developed for functional automation suites, and hence, this is one of the major differences.

For the most realistic results, conventional wisdom is to test automation on actual devices—without optical recognition, emulation, jailbreaking, or tethering. It is impractical to try to automate everything, especially for mobile devices. However, leveraging commercial off-the-shelf (COTS) tools can vastly reduce the cost of automation and thereby enhance the benefits of the automation process.

In the following section, we will discuss in detail the challenges that make mobile automation vastly different from conventional functional automation.
The following are some of the issues that make the effective testing automation of mobile applications challenging:

- **Restricted access to native methods to enable automation tools**: Traditional functional automation tools utilize native operating system methods to emulate user interactions. This is comparatively easy to do as the operating system allows access. However, the same level of access is not available with a mobile operating system. Also, inter-application interactions are restricted in a mobile operating system and each application is treated as an individual thread. This is normally only allowed when a phone is rooted or when the application under test is modified to allow instrumentation access. So, using other software (the test automation tool) to control user inputs in a mobile application is much more difficult to achieve and consequently slower or more error prone.

  For example, if an Android application under test makes a call to the photo gallery, then the automated test would not be able to continue because a new application comes to the foreground.

- **Lack of prediction techniques for UI synchronization in a Mobile environment**: In addition to the restricted access mentioned in the previous point, mobile application user interface response times are dependent on many variables, such as connection speed and device configuration other than the server response times. Hence, it is much harder to predict the synchronization response in a mobile application. Due to this automation of mobile, the application is more prone to be unstable unless hardcoded wait times are included in the automation scripts.

- **Handling location-specific changes in the application behavior**: Many mobile applications are designed to interact with the user location, and behave differently as per the change in GPS coordinates. Since network strengths cannot be controlled externally, it is very difficult to predict the application behavior and to replicate the preconditions of a network strength-specific use case through the use of automation. So, this is another aspect that every automation solution has to address appropriately. Some automation tools allow the simulation of such network conditions that should be specifically handled while developing the automation suite.
• **Supporting application behavior changes for varied form factors**: As explained earlier, since there are different screen sizes, available for mobile devices, the behavior of the application is often specific to the screen size owing to responsive design techniques that are now quite widely used. Even with the change in the orientation of the devices, application use cases have alternative behavior. For example, an application interface loaded in the portrait mode would appear different, with objects in different locations than they would appear in the landscape mode. Hence, automation solutions would need to factor this in and ensure that such changes are handled in a robust and scalable way.

• **Scripting complexity due to diversity in OS**: Since many applications are developed to support various OSes, especially mobile web applications, it is a key challenge to handle application differences, such as mobile device input methods for various devices, as devices differ in keystrokes, input methods, menu structures, and display properties. With different mobile operating systems in the market, such as Android, iOS, Brew Symbian, Tizen, Windows, and BlackBerry (RIM), each having its own limitations and variations, creation of a single script for every device is a challenge that needs to be adequately tackled in order to make the automation solution more robust, maintainable, and scalable to support newer devices in future.

In subsequent chapters, we will discuss how to effectively manage these using various techniques.

## Mobile application packages

With the advancement in wireless technology, big technology companies, such as Apple, Amazon, Google, and so on, came out with a solution that provides users with a more realistic approach to finding information, making decisions, shopping, and other countless things at their fingertips by developing mobile applications for their products. The main purpose of developing mobile applications was actually to retrieve information using various productivity tools, which includes calculator, e-mail, calendar, contacts, and many more. However, with more demand for and the availability of resources, there was a rapid growth and expansion in other categories, such as mobile games, shopping, GPS and location-based services, banking, order tracking, ticket purchases, and recently, mobile medical applications.
The distribution platforms, such as Apple App Store, Google Play, Windows Phone Store, Nokia Store, and BlackBerry Application World, are operated by the owners of the mobile operating systems, and mobile applications are made available to users by them. We usually hear about the terms such as a native application, hybrid application, or web application, so, did you ever wonder what they are and what is the difference between them? Moving ahead, we will discuss the different mobile packages available for use and their salient features that make an impact on the selection of a strategy and testing tool for automation.

The different mobile packages available are:

- Native applications
- Web applications
- Hybrid applications

**Native applications**

Any mobile application needs to be installed through various distribution systems, such as Application Store and Google Play. Native applications are the applications developed specifically for one platform, such as iOS, Android, Windows, and many more. They can interact and take full advantage of operating system features and other software that is typically installed on that platform. They have the ability to use device-specific hardware and software, such as the GPS, compass, camera, contact book, and so on.

These types of applications can also incorporate gestures such as standard operating system gestures or new application-defined gestures. Native applications have their entire code developed for a particular operating system and hence have no reusability across operating systems. A native application for iOS would thus have its application handles built specifically for Objective-C or Swift and hence would not work on an Android device. If the same application needs to be used across different operating systems, which is a very logical requirement for any successful application, then developers would have to write a whole new repository of code for another mobile operating system.

This makes the application maintenance cumbersome and the uniformity of features is another challenge that becomes difficult to manage. However, having different code bases for different operating systems allows the flexibility to have operating-system-specific customizations that are easy to build and deploy. Also, today, there is a need to follow very strict "look and feel" guidelines for each operating system. Using a native application might be the best way to keep this presentation correct one for each OS.
Also, testing native applications is usually limited to the operating system in question and hence, the fragmentation is usually limited in impact. Only manufactures and operating system versions need to be considered.

Mobile web applications

A mobile web application is actually not an application but in essence only websites that are accessed via a mobile interface, and it has design features specific to the smaller screen interface and it has user interactions such as swipe, scroll, pinch, and zoom built in. These mobile web applications are accessed via a mobile browser and are typically developed using HTML or HTML5. Users first access them as they would access any web page. They navigate to a special URL and then have the option of installing them on their home screen by creating a bookmark for that page.

So, in many ways, a web application is hard to differentiate from a native application, as in mobile screens, usually there are no visible browser buttons or bars, although it runs in mobile browsers. A user can perform various native application functionalities, such as swiping to move on to new sections of the application.

Most of the native application features are available in the HTML5 web application, for example, they can use the tap-to-call feature, GPS, compass, camera, contact book, and so on. However, there are still some native features that are inaccessible (at least for now) in a browser, such as the push notifications, running an application in the background, accelerometer information (other than detecting landscape or portrait orientations), complex gestures, and more.

While web applications are generally very quick to develop with a lot of ready-to-use libraries and tools, such as AngularJS, Sencha, and JQuery, and also provide a unique code base for all operating systems, there is an added complexity of testing that adds to the fragmentation problem discussed earlier. There is no dearth of good mobile browsers and on a mobile device, there is very limited control that application developers can have, so users are free to use any mobile browser of their choice, such as Chrome, Safari, UC Browser, Opera Mobile, Opera Mini, Firefox, and many more. Consequently, these applications are generally development-light and testing-heavy. Hence, while developing automation scripts, the solution has to consider this impact, and the tool and technique selected should have the facility to run scripts on all these different browsers.

Of course, it could be argued that many applications (native or otherwise) do not take advantage of the extra features provided by native applications. However, if an application really requires native features, you will have to create a native application or, at least, a hybrid application.
Hybrid applications

Hybrid applications are combinations of both native applications and web applications, and because of that, many people incorrectly call them web applications. Like native applications, they are installed in a device through an Application Store and can take advantage of the many device features available. Just like web applications, hybrid applications are dependent on HTML being rendered in a browser, with the caveat that the browser is embedded within the application. So, for an existing web page, companies build hybrid applications as wrappers without spending significant effort and resources, and they can get their existence known in Application Store and have a star rating! Web applications usually do not have one and hence have this added disadvantage of lacking the automatic publicity that a five-star rating provides in the mobile stores.

Because of cross-platform development and significantly low development costs, hybrid applications are becoming popular, as the same HTML code components are reusable on different mobile operating systems. The other added advantage is that hybrid applications can have the same code base wrapped inside an operating-system-specific shell thereby making it development-light. By removing the problem posed by various device browsers, hybrid applications can be more tightly controlled, making them less prone to fragmentation, at least on the browser side. However, since they are hybrid applications, any automation testing solution should have the ability to test across different operating system and version combinations, with the ability to differentiate between various operating-system-specific functionality differences. Various tools such as PhoneGap and Sencha allow developers to code and design an application across various platforms just by using the power of HTML.

Factors to be considered during mobile testing

In many aspects, an approach to perform any type of testing is not so different from mobile automation testing. From methodology and experience, while working with the actual testing tools, what testers have learned in testing can be applied to mobile automation testing.

So, a question might come to your minds that then, where does the difference lie and how should you accommodate these differences? So, following this topic, we will see some of the factors that are highly relevant to mobile automation testing and require particular attention, but if handled correctly, then we can ensure a successful mobile testing effort.
Some of the factors that need to be taken care of in testing mobile applications are as follows:

- **Testing for cross device and platform coverage:** It is not feasible to test an application on each and every available device because of the plethora of devices that support the application across different platforms, which means you have to strategically choose only a limited, but sufficient set of physical devices. You need to remember that testing on one device, irrespective of whether it is of the same make, same operating system version, or uses the same platform cannot ensure that it would work on any other device. So, it is important that, at the very least, most of the critical features, if not all, are tested on a physical device. Otherwise, the application always runs a risk of potential failure on an untested device, especially when the target audience for the application is widespread, such as for a game or banking application.

Use of emulated devices is one of the common ways to overcome the issues of testing on numerous physical devices. Although this approach is generally less expensive, we cannot rely completely on the emulated devices for the results they present, and with emulators, it may be quite possible that test conditions are not close enough to the real-life scenarios.

So, an adequate coverage of different physical devices is required to test these following variations, providing sufficient coverage in order to negate the effects of fragmentation and have sufficient representation of these various factors:

- Varying screen sizes
- Different form factors
- Different pixel densities and resolutions
- Different input methods, such as QWERTY, touch screen, and more
- Different user input methods, such as swipes, gestures, scrolling, and many more
• **Testing different versions of an operating system of the same platform:** For thorough testing, we need to test the application on all major platforms, such as Android, iOS, Windows, and others, for the target customer base, but each one of them has numerous versions available that keep on growing regularly. Most commonly, testing automation on the latest version of any operating system can be sufficient, as the operating systems are generally backward compatible. However, due to fragmentation of the Android OS, the application would still need to be tested on at least the most commonly used versions besides the latest ones, which in some cases may be significantly behind the latest version. This is because there may be many Android devices that are on an earlier version of Android and are not supported by the latest versions of Android.

• **Testing of various network types and network providers:** Most of the mobile applications, such as banking- or information-search-related applications require network connectivity, such as CDMA or GSM, at least partially, if not completely. If the application talks to a server about the flow of information to and fro, testing on various (at least all major) network providers is important. The network infrastructure used by network providers may affect data communication between application and the backend. Apart from the different network providers, an application needs to be tested on other modes of network communication, such as Wi-Fi network as well.

• **Testing for mobile-environment-specific constraints:** The mobile environment is very dynamic and has constraints, such as limited computing resources, available memory, in-between calls or messages, network switching, battery life, and a lot of other sensors and features, such as accelerometer, gyroscope, GPS, memory cards, camera, and others, present in the device, as an application's behavior depends on these factors. An application should integrate or interact (if required) with these features gracefully, and sufficient testing needs to be carried out in various situations to ensure this. However, oftentimes, it is not practically feasible to recreate all permutations and combinations of these factors, and hence a strategic approach needs to be taken to ensure sufficient coverage.
Ensuring Five-star Rating in the MarketPlace

• **Testing for the unpredictability of a mobile user:** A tester has to be more cautious and should expand the horizon while testing the applications. They should make sure that an application provides an overall good response to all users and a good user experience; hence, **User Experience (UX)** testing invariably needs to be performed to a certain degree for all mobile applications. Since, a mobile application’s audience comprises of various people ranging from nontech people to skilled technical users and from children to middle-aged users. Each of the users have their own style of using the application and have set their own expectations of it. A middle-aged or an aged user will be much calmer while using any application than someone who is young when it comes to the performance of the application. In general, we can say that mobile users have set incredibly high expectations of the applications available in the marketplace.

In the subsequent chapters, we will develop our understanding of how to overcome these challenges during automation design and how to strategize effectively. However, before that, we need to be aware of the different approaches for mobile test automation.

**Mobile automation testing approaches**

In this section, you will understand the different approaches used for automation of a mobile application and their salient points.

There are, broadly speaking, four different approaches or techniques available for mobile application testing automation:

• Test automation using physically present real devices
• Test automation using emulators and simulators
• Mobile web application test automation through the user agent simulation technique
• Cloud-solutions-based test automation
Automation using real devices

As the name suggests, this technique is based on the usage of real devices that are physically present with the testing automation team. Since this technique is based on the usage of real devices, it is a natural consequence that the Application Under Test (AUT) is also tested over a real network (GSM, CDMA, or Wi-Fi). To establish connectivity of the automation tool with the devices, any of the communication mechanisms, such as USB, Bluetooth, or Wi-Fi can be used; however, the most commonly used and the most reliable one is the USB connection. After the connection is established between the machines on which the automation tool is installed and the Device Under Test (DUT), the automation scripts can capture object properties of the AUT and later, the developed scripts can be executed on other devices as well, but with minor modifications.

There are numerous automation tools, both licensed as well as open source freeware, available for mobile automation. Some commonly used licensed tools are:

- Experitest SeeTest
- TestPlant eggPlant Mobile /eggOn
- Jamo Solutions M-eux Test
- ZAP-fiX

Prominent tools for Android and iOS automation are:

- Selenium with solutions such as Selendroid and Appium along with iOS and Android drivers
- MonkeyTalk (formerly FoneMonkey)

The following are the salient features of this approach:

- The AUT is accessed on devices either by using a real mobile network or Wi-Fi network and can also be accessed by the Intranet network of the machine to which it is connected
- The automation testing tool is installed on the desktop that uses the USB or Wi-Fi connectivity to control devices under test
Steps to set up automation

For automation on real devices, scripts are required to be executed on the devices with a USB or Wi-Fi connection to send commands via the execution tool to the devices. The following is a step-by-step description of how to perform the automation on real devices:

1. Determine the device connectivity solution (USB or Wi-Fi connectivity) based on the available setup. In some cases, USB connectivity is not enabled due to security policies and only in those cases is a Wi-Fi connection utilized.

2. Identify the tool to be used for the automation based on the tool feasibility study of the application. We will discuss in detail how to conduct a mobile automation tool feasibility study and the parameters that should be considered in subsequent chapters, when we discuss this technique in detail.

3. Procure the required licenses (seat or concurrent) if a licensed tool is selected. License procurement might mean that lengthy agreements need to be signed by both parties, besides arranging for the payment of services such as support. So, this step should be well planned with enough buffer time.

4. If the existing automation setup is to be leveraged, then an additional license needs to be acquired that corresponds to the tool (such as Quick Test Professional, Quality Center, and more). In some cases, you might also have to integrate the existing automation scripts developed with tools such as Quick Test Professional/Unified Functional Testing along with the automation scripts developed for the mobile. In such a case, the framework already in place needs to be modified.

5. Install the tools on the automation computer and establish the connectivity with the real devices. Installation may not be as simple as just running an executable file when it comes to mobile automation. There are various network-level settings and additional drivers that are needed to connect the computer and to control various mobile devices from the computer. Hence, all this should be done and planned well in advance.

6. Script the test cases and execute them on real devices.

Limitations of this automation

This approach has the following limitations:

- The overall cost can be high as multiple devices are required to be procured for different teams and testers
- Maintenance and physical security can be an overhead
• Script maintenance can be delayed if testing cycles are overlapping with functional and automation teams

Emulators-based automation

Emulators are programs that replicate the behavior of a mobile operating system and, to some extent, the device features on a computer. So, in essence, these programs are used to create virtual devices. So, any mobile application can be deployed on such virtual devices and then tested without the use of a real device. Ideally speaking, there are two types of mobile device virtualization programs: emulators and simulators.

From a purely theoretical standpoint, the following are the differences between an emulator and a simulator.

A device emulator is a desktop application that emulates both the mobile device hardware and its operating systems; thus, it allows us to test the applications to a lesser degree of tolerance and better accuracy. There are also operating system emulators that don’t represent any real device hardware, but rather the operating system as a whole. These exist for Windows Mobile and Android, but a simulator is a simpler application that simulates some of the behavior of a device, does not emulate hardware, and does not work over the real operating system. These tools are simpler and less useful than emulators. A simulator may be created by the device manufacturer or by some other company that offers a simulation environment for developers. Thus, simulator programs have lesser accuracy than emulator programs.

For the sake of keeping the discussion simple, we will refer to both as emulators in this chapter and in later chapters; when we discuss this technique in detail, we will refer to each of them individually.

Since this technique does not use real devices, it is a natural consequence that the AUT is not tested over a real network (GSM, CDMA, or Wi-Fi), and the network connection of the machine is utilized to make a connection with the application server (if it connects to a server, which around 90 percent of mobile applications do). Since the virtual devices are available on the computer, there is no external connection required between the device’s operating system and automation tool. However, an emulator is not as simple as automating any other program because the actual AUT runs inside the shell of the virtual device. So, a special configuration needs to be enabled with the automation tools to enable the automation on the virtual device.
Ensuring Five-star Rating in the MarketPlace

The following is a diagram depicting an Android emulator running on a Windows 7 computer:

In most projects, this technique is used for prelaunch testing of the application, but there are cases where emulators are automated to a great extent. However, since the emulator is essentially more limited in scope than the real devices, mobile-network-specific and certain other features such as memory utilization cannot be relied upon while testing automation with emulators. There are numerous automation tools, both licensed as well as of an open source freeware available for mobile automation on these virtual devices, and ideally, emulators for various mobile platforms can be automated with most of the tools that support real device automation.

The prominent licensed tools are:

- ExperiTest SeeTest
- TestPlant eggPlant Mobile /eggOn
- Jamo Solutions M-eux Test

Tools such as Selenium and ExperiTest SeeTest can be used to launch device platform emulators and execute scripts on the AUT.
The prominent free-to-use tools for emulator automation are:

- Selenium WebDriver
- Appium
- MonkeyTalk (formerly FoneMonkey)

Since emulators are also software that run on other machines, device-specific configurations need to be performed prior to test automation and have to be handled in the scripts. The following is the conceptual depiction of this technique.

The emulator and simulator programs are installed on a computer with a given operating system, such as Windows, Linux, or Mac, which then virtualizes the mobile operating system, such as Android, iOS, RIM, or Windows, and subsequently, which can be used to run scripts that emulate the behavior of an application on the real devices.

**Steps to set up automation**

The following are the steps to set up the automation process for this approach:

1. Identify the various platforms for which the AUT needs to be automated.
2. Establish the connectivity to AUT by enabling the firewall access in the required network for mobile applications.
3. Identify the various devices, platforms, emulators, and device configurations, according to which test needs to be carried out.
4. Install emulators/simulators for the various platforms.
5. Create scripts and execute them across multiple emulators/simulators.

**Advantages**

This approach has the following advantages:

- Standalone emulators that don't have real devices can be utilized
- No additional connectivity is required for automation
- This provides support for iOS and Android with freeware
- This provides support to all platforms and types of applications with licensed tools, such as Jamo Solutions M-eux and ExperiTest SeeTest
Limitations

This approach has the following limitations:

- This can be difficult to automate as the emulators and simulators are themselves not thoroughly tested software and might have unknown bugs.
- Selenium WebDriver cannot be used to automate Android applications in some versions due to a bug in the Android emulator.
- It might sometimes be difficult to triangulate a defect that is detected on a virtual device and it might be needed that you recreate it on a real device first. In many cases, it has been observed that defects caught on emulators are not reproduced on real devices.
- For iOS simulators, access to a Mac machine with Xcode is required, which can be difficult to set up in a secure Offshore Development Center (ODC) due to security restrictions.

User agent-simulation-based automation

The third technique is the simplest of all. However, it is also very limited in its scope of applicability. It can be used only for mobile web applications and only to a very limited extent. Hence, it is generally only used to automate the functional regression testing of mobile web applications and rarely used for GUI validations.

User agent is the string that web servers use to identify information, such as the operating system of the requester and the browser that is accessing it. This string is normally sent with the HTTP/HTTPS request to identify the requester details to the server. Based on this information, a server presents the required interface to the requesting browser.

This approach utilizes the browser user agent manipulation technique. This is depicted in the following schematic diagram:
In this approach, an external program or a browser add-on is used to override the user agent information that is sent to the web application server to identify the requestor system as a mobile instead of its real information. So, for example, when a web application URL such as https://www.yahoo.com is accessed from a mobile device, the application server detects the requester to be a mobile device and redirects it to https://mobile.yahoo.com/, thereby presenting the mobile view. If the user agent information is overridden to indicate that it is coming from a Safari browser on an iPhone, then it will be presented with the mobile view.
Ensuring Five-star Rating in the MarketPlace

The following screenshot depicts how the application server has responded to a request when it detects that the request is from an iPhone 4 and is presented the mobile view:

Since the mobile web application is accessed entirely from the computer, automation can be done using traditional web browser automation tools, such as Quick Test Professional/Unified Functional Testing or Selenium.

The salient features of this technique are as follows:

• With browser user agent manipulation, any mobile platform can be simulated
• Browser user agent manipulation is limited to only mobile web applications and is not extended to native and hybrid applications
• Browser simulation can be done using freeware files that are available for all leading web browsers
• The common user agent switching tools are:
  - Bayden UAPick for IE
  - User agent switcher add-on for Firefox
  - Fiddler for IE
  - Modify Headers for Firefox
  - UA Spoof add-on for Chrome
  - Built-in device emulator with Chrome that can be accessed from developer tools

**Steps to set up the automation**
The following are the steps to set up the automation process for this approach:

1. Identify the various platforms for which the AUT needs to be validated.
2. Identify the user-agent switcher tool that corresponds to any browser that needs to be leveraged for testing.
3. Identify the user-agent string for all platforms in scope and set up configuration in the user-agent switcher tool.
4. Leverage any functional testing tool that offers testing capabilities using any web browser, for example, Quick Test Professional, RFT, SilkTest, and Selenium WebDriver.

**Advantages**
This approach has the following advantages:

• All platforms can be automated with little modification of scripts
• Quick implementation of automation solution
• Can leverage an open source software, such as Selenium for automation
• Existing automation set up can be leveraged
Limitations
This approach has the following limitations:

• Least representative of real device-based tests
• Device-specific issues cannot be captured through this approach
• This cannot be used for UI-related test cases
• This approach supports only web-based mobile applications

Cloud-based automation
This technique provides most of the capabilities for test automation, but is also one of the more expensive techniques. In this technique, automation is done on real devices connected to real networks that are accessed remotely through cloud-based solutions, such as Perfecto Mobile, Mobile Labs, Sauce Labs, and DeviceAnywhere.

The salient features of this technique are as follows:

• Cloud-based tools, such as Perfecto mobile and Device Anywhere provide a WYSIWYG (What You See Is What You Get) solution for automation
• Both OCR (Optical Character Recognition) and native object recognition and analysis is utilized in these tools for automation
• These tools also provide simple high-level keywords, such as Launch Browser, Call Me, and many more that can be used to design test cases
• The scripts that are thus created need to be re-recorded for every new type of device due to differences between the interface and GUI objects
• Mobile devices are accessed via a web interface or thick-client, by teams in various regions
• The devices are connected to real networks that use Wi-Fi or various mobile network operators (AT&T, Vodafone, and more)
• The AUT is accessed via the Internet or through a secure intranet connection
• This approach provides offer integration with common automation tools such as Quick Test Professional/UFT and Selenium
Steps to set up the automation

The following are the steps to set up the automation process for this approach:

1. Identify the various platforms and devices for which the AUT needs to be automated.
2. Establish connectivity to AUT by enabling the firewall access for mobile web applications.
3. Open an account with the chosen cloud solution provider and negotiate to get the licenses for automation or set up a private cloud infrastructure within your company premises.
4. Install the cloud service provider client-side software setup along with the automation plugin for the relevant tool of choice (UFT or Selenium).
5. Book the devices as per testing needs (this usage normally has a cost associated with it).
6. Create scripts and execute across multiple devices.

Advantages

This approach has the following advantages:

- This allows us to test automation on multiple devices of various manufactures (hardware)
  For example: Samsung, Apple, Sony, Nokia
- A script can be executed on multiple mobile devices from the same manufactures (models)
  For example: Galaxy SII, Galaxy SIII, iPhone 4S, iPhone 5, iPad2
- Scripts can be tested on different platforms (software)
  For example: Android 2.3 - 4.4, iOS 4-8, Symbian, Bada

Limitations

This approach has the following limitations:

- Network latency may be experienced
- Cost can be high as fees depends on device usage
- Setting up a private mobile lab is costly, but may be necessary due to an organization's security policies, particularly in legally regulated industries, such as BFSI organizations
Ensuring Five-star Rating in the MarketPlace

Types of mobile application tests
Apartment from the usual functional test, which ensures that the application is working as per the requirements, there are a few more types that need to be handled with an automation solution:

- **Interrupt testing**: A mobile application while functioning may face several interruptions that can affect the performance or functionality of an application. The different types of interruptions that can adversely affect the functionality of an application are:
  - Incoming calls and SMS or MMS
  - Receiving notifications, such as **Push Notifications**
  - Sudden removal of battery
  - Transfer of data through a data cable by inserting or removing the data cable
  - Network/Data loss or recovery
  - Turning a Media Player off or on

Ideally, an application should be able to handle these interruptions, for example, whenever an interruption is there, an application can go into a suspended state and resuming afterwards. So, we should design automation scripts in such a way that they can not only test these interrupts, but they can reliably also reproduce them at the requisite step of the flow.

- **UI testing**: A user interface for a mobile application is designed to support various screen sizes and hence, the various components of a mobile application screen appear differently or in some cases, even behave differently as per the OS or device make. Hence, any automation script needs be able to work with varying components and also be able to verify the component's behavior. Use of automation ensures that the application is quickly tested and the fixes are regression tested across different applications. Since UI is where the end users interact with the application, use of a robust automation suite is the best way to ensure that the application is thoroughly tested so that it rolls out to the users in the most cost-effective manner. A properly tested application makes the end user experience more seamless and thereby, the application under test is more likely to get a better star rating and its key to commercial success.
**Installation testing**: Installation testing ensures that the installation process goes smoothly without the user facing any difficulty. This type of a testing process includes not only installing an application but also updating and uninstalling an application. Use of automation to install and uninstall any application as per the defined process is one of the most cost-effective ways to do this type of testing.

**Form factor testing**: Applications may behave differently (especially in terms of user interface) on smartphones and tablets. If the test application supports both smartphones and tablets, it should be tested on both form factors. This can be treated as an extension to the UI testing type.

### Selection of the best mobile testing approach

While selecting a suitable mobile testing approach, you need to look at the following important considerations:

- **Availability of automation tools**: The availability of relevant mobile automation tool plays a big role in the selection and implementation of the mobile automation approach.

- **Mode of connection of devices**: This is one of the primary, if not the most important, aspect that plays a pivotal role in the selection of a mobile automation approach.

There are different ways in which devices can be connected to the automation tools such as:

- Using a USB connection
- Using a Wi-Fi connection
- Using a Bluetooth connectivity (only for a very limited set of tools)
- Using localized hotspots, that is, having one device as a hotspot and other devices riding its network for access
- Cloud connection
- Use of emulators and simulators
Ensuring Five-star Rating in the MarketPlace

All these approaches need specific configurations on machines, and with the automation tools, which may sometimes be restricted, any automation solution should be able to work around the constraints in various setups. The key consideration is the 

degree of tolerance of the automation solution.
The four different approaches that we discussed earlier in this chapter have each got a different level of accuracy. The least accurate is the user agent-based approach because it relies just on a web browser's rendering on a Windows machine rather than a real device. The most accurate approach, in terms of closeness to the real-world situation, is the use of real devices. However, this approach suffers from restrictions in terms of scalability of the solution, that is, supporting multiple devices simultaneously. Use of emulators and simulators is also prone to inaccuracies with respect to the real-device features, such as RAM, screen resolutions, pixel sizes, and many more. While working with cloud-based solutions, a remote connection is established with the devices, but there can be unwanted signal delays and screen refresh issues due to network bandwidth issues.

So, any approach that is selected for automation should factor in the degree of tolerance that is acceptable with any automation suite. For example, for a mobile application that makes heavy usage of graphics and advanced HTML 5 controls, such as embedded videos and music, automation should not be carried out with an emulator solution, as the degree of accuracy would suffer adversely and usually beyond the acceptable tolerance limit.

Consider another application that is a simple mobile web application with no complex controls and that doesn't rely on any mobile-device-specific controls, such as camera controls, or touch screen sensitive controls, such as pinch and zoom. Such an application can easily be automated with the user agent-based approach without any significant impact on the degree of accuracy.

If an application uses network bandwidth very heavily, then it is not recommended to use the cloud-based approach, as it will suffer from network issues more severely and would have unhandled exceptions in the automation suite. Conversely, the cloud-based approach is most suitable for organizations that have geographically and logically dispersed teams that can use remotely connected devices from a single web interface. This approach is also very suitable when there are restrictions on the usage of other device connection approaches, such as USB, Wi-Fi, or Bluetooth. Although this approach does need additional tools to enable cloud access, it is a worthwhile investment for organizations that have a high need for system and network security, such as banking and financial organizations.
Troubleshooting and best practices

The following best practices should ideally be followed for any mobile automation project:

- The mode of connectivity between the AUT, DUT, and computer on which the automation tool is installed should be clearly established with all the considerations of any organization's security policies. In most cases, there is no way to work around to the absence of USB connectivity, other than to use cloud-based automation solutions. So, before starting a project, the physical setup should be thoroughly vetted.

- The various operating systems and versions, mobile equipment manufacturers, and different form factors that need to be supported with the application, and consequently, the automation solution should be designed to support all of them. However, if you start automating before identifying all the supported devices, then there would invariably be a lot of rework required to make the scripts work with other devices. Hence, automation scripts should be made for all supported OSes and devices right from the design stage.

- A user agent-based automation can only be implemented for mobile web applications. It is a cost-effective and quick way to implement solutions since it involves automation of just a few basic features. However, this technique should not be relied upon for validating GUI components and should always be accompanied with a round of device testing.

- If any simulation or emulation technique (user agent or emulators/simulators) is used for automation, then it should strictly be used for functional regression testing on different device configurations. Ideally, projects utilizing these solutions should also have a GUI testing round with real devices, at least for the first release.

- If a geographically-distributed team is to utilize the automation solution, for example, an offshore-onsite team that needs to use the same devices, then the most cost-effective solution in the long run is the cloud-based automation. Even though the initial setup cost of the cloud solution generally is the highest of the four techniques, since different teams can multiplex and use devices from different locations and so the overall cost is offset by using fewer devices overall.
Ensuring Five-star Rating in the MarketPlace

- During the use of emulators/simulators, the automation scripts should be designed to trigger the virtualization program with the required settings for memory, RAM, and the requisite version of the operating system, so that there is no manual intervention required to start the programs before you trigger the execution. Also, this way, scripts can be triggered remotely and in an unmonitored way.
- Irrespective of the technique utilized, a proper framework should be implemented with the automation solution. In the next chapter, we will discuss the various automation frameworks and their specific differences for mobile automation solutions.

Summary
In this chapter, we learned what mobile test automation is, what are the different mobile packages that are available, and what factors should be considered during mobile automation testing. We then moved on to learn the different types of approaches and selection of the best approach according to any specific project requirements. So, it is evident that with the use of automation to test any mobile application, a good user experience can be ensured with a defect-free software, with which a good star rating can be expected for the AUT. In the next chapter, we will understand techniques used to design various mobile automation frameworks.
Where to buy this book

You can buy Mastering Mobile Test Automation from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.