A COMPARATIVE STUDY ON AUTOMATED ANDROID APPLICATION TESTING TOOLS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜLÇİN HÖKELEKLİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

MAY 2016

A COMPERATIVE STUDY ON ANDROID APPLICATION TESTING TOOLS

Submitted by **GÜLÇİN HÖKELEKLİ** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Nazife Baykal _____
Director, Informatics Institute

Prof. Dr. Yasemin Yardımcı Çetin _____
Head of Department, Information Systems

Assoc. Prof. Dr. Aysu Betin Can _____
Supervisor, Information Systems, METU

Examining Committee Members

Assoc. Prof. Dr. Aysu Betin Can
IS, Middle East Technical University _____

Assoc. Prof. Dr. Altan Koçyiğit
IS, Middle East Technical University _____

Assist. Prof. Dr. Erhan Eren
IS, Middle East Technical University _____

Assist. Prof. Dr. Murat Perit Çakır
COGS, Middle East Technical University _____

Assist. Prof. Dr. Abdül Kadir Görür
CENG, Çankaya University _____

Date: 06.05.2016

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Gülçin HÖKELEKLİ

Signature:

ABSTRACT

A COMPARATIVE STUDY ON AUTOMATED ANDROID APPLICATION TESTING TOOLS

Hökelekli, Gülçin

M.Sc., Department of Information Systems

Advisor: Assoc. Prof. Dr. Aysu Betin Can

MAY 2016, 86 Pages

Nowadays, as mobile devices have become widespread, mobile application development has become an area which is considerably popular. This popularity increases the importance of mobile application testing. Distinguishing properties of mobile devices increase the importance of test automation. Thus, the number of mobile test automation tools is growing. Each tool has some advantages and limitations. The aim of this study is to compare the most popular mobile testing tools. We choose Android testing tools because of Android's prevalence in the market. To achieve this aim of the study, firstly we have identified the criteria that will be used for comparison of the testing tools. Then, we have selected three most commonly used Android test automation tools. Afterwards, we have analyzed and compared the selected tools in terms of the criteria identified before. In order to make this comparison we have made a detailed research on tutorials and conducted a case study in which we have written and run same test cases by using each of the selected tools.

Keywords: Android Application Testing, Test Automation, Mobile Application Testing, Android Test Automation Tools

ÖZ

ANDROID UYGULAMALARI TEST OTOMASYON ARAÇLARI ÜZERİNE KARŞILAŞTIRMALI ÇALIŞMA

Hökelekli, Gülçin

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Danışmanları: Doç. Dr. Aysu Betin Can

Mayıs 2016, 86 Sayfa

Günümüzde mobil cihazların yaygınlaşmasıyla mobil uygulama geliştirme önemli ölçüde popüler hale gelmiştir. Bu popülarite mobil uygulama testlerinin önemini arttırmaktadır. Mobil cihazların ayırt edici özellikleri, test otomasyonunun önemini arttırmaktadır. Bu nedenle, mobil uygulama test otomasyon araçlarının sayısı artmaktadır. Her bir araç, bazı avantajlara ve kısıtlara sahiptir. Bu çalışmanın amacı, en popüler mobil test araçlarını karşılaştırmaktır. Android'in pazardaki yaygınlığından dolayı Android test araçları seçilmiştir. Çalışmamızın amacına ulaşabilmek için ilk olarak test araçlarını karşılaştırırken kullanacağımız kriterleri belirledik. Sonra, en yaygın olarak kullanılan Android test otomasyon araçlarını seçtik. Daha sonra, seçtiğimiz araçları analiz ettik ve belirlemiş olduğumuz kriterlere göre karşılaştırdık. Bu karşılaştırmayı yapmak için dokümanlar üzerinde detaylı bir araştırma yaptık ve seçmiş olduğumuz her bir araç ile aynı test senaryolarını yazdığımız bir örnek olay incelemesi yaptık.

Anahtar Kelimeler: Android Uygulama Testi, Test Otomasyonu, Mobil Uygulama Testi, Android Test Otomasyon Aracı

To My Family
Sema, Ruhi and Furkan

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Assoc. Prof. Dr. Aysu BETİN CAN for her guidance, encouragements and support during my thesis study.

I also would like to thank my amazing family: Sema, Ruhi and Furkan HÖKELEKLİ for their concern, patience and encouragements during this process.

I would like to send special thanks to my best friend Merve Vildan ŞİMŞEK for all of her help, endless support and motivation throughout Master's Program.

I am also grateful to my colleagues and managers at Turkish Ministry of Economy Department of IT for their support and understanding during my Master's Program.

I would also like to thank TÜBİTAK for scholarship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IEEE	Institute of Electrical and Electronics Engineers
IT	Information Technology
SDK	Software Development Kit
IDE	Integrated Development Environment
UI	User Interface
GUI	Graphical User Interface
APP	Application
JDK	Java Development Kit
RAM	Random Access Memory
CPU	Central Processing Unit
LOC	Line of Code
APK	Android Application Package

CHAPTER I

1. INTRODUCTION

Since the first decade of the twenty-first century, there is a dramatic explosion on mobile communication technologies. By 2007, when Apple Inc. introduced IPhone which is “one of the first smartphones to use a multi-touch interface” [1], mobile phones started to be replaced with smartphones. A smartphone is actually a mobile phone that has an operating system, a more powerful hardware and personal computer functionalities. As smartphones have become an essential part of our lives, the number of mobile applications is growing in a remarkable way. By March 2016, there are 2,051,820 Android applications in the market [2]. The world’s changing its direction to mobile, makes mobile application development a major area in IT industry.

The increase in the importance of mobile application development makes mobile application testing very significant as well. IEEE Computer Society defines mobile application testing as: "testing activities for native and Web applications on mobile devices using well-defined software test methods and tools to ensure quality in functions, behaviors, performance, and quality of service, as well as features, such as mobility, usability, interoperability, connectivity, security, and privacy.”[3]

As traditional software testing, mobile application testing can be conducted using either manual or automated approach. In automated testing, special software are used for test case generation, execution and verification while in manual testing those processes are done by human [4]. The significance of mobile application testing raises the need for test automation frameworks for mobile applications. Thus, software companies have developed various mobile application testing frameworks with different capabilities and properties.

The aim of this study is to make a comparison between most widely used mobile application testing tools. In this comparison, we examined the detailed technical insight about selected tools. Because of Android’s prevalence in the market, we choose Android application testing tools.

Our study has three parts. Firstly, we have identified criteria to be used for comparison. We have aimed to make a detailed list of criteria to present the technical capabilities of the tools effectively. In order to accomplish this aim, we have used mobile application testing checklists and our exploratory study together. We have

focused on GUI based functional testing, interrupt testing and memory leakage testing [5]. In the second part, we have evaluated selected tools; Robotium, Appium and UIAutomator with respect to the criteria we have identified. In the last part, we have conducted a case study. We have selected two mobile applications to implement and run same test cases using each tool.

There are a number of studies on mobile testing frameworks and making comparison between them. Gunasekaran and Bargavi [6] reviewed five testing tools; Appium, MonkeyTalk, Ranorex, Robotium and UIAutomator. Their study includes main information about capabilities and implementation of these tools and a comparative study. They used 6 criteria including high level properties. These criteria are; supporting Android or iOS platforms, scripting language, being cross platform, ability to take screenshot and ability to verify expected and actual outputs. They did not consider properties such as connectivity, simulating gestures or interruptions.

Gao,Bai,Tsai and Uehara [3] gives general information about mobile testing automation such as testing types, approaches and difficulties of mobile application testing. Their study also includes a table of comparison of mobile testing tools. They focused on 15 mobile testing tools including the tools that support iOS platform. However, their criteria are also very high level. The attributes they focused on are; testing types provided by the tools, supported platforms, supported application types, ability to test on emulator or real device, scripting languages, ability to record and replay, being open source and requiring subscription. Even the number of tools and criteria are not small, they did not give information such as interaction of soft keyboard and notification area, ability to test APK files, or ability to run tests using command prompt.

Singh, Gadgil and Chudgor [7] conducted a study about the mobile testing tool; Appium. They gave detailed information about Appium. They added a comparison table as well. Their table includes six mobile testing tools: Instruments, UIAutomator, Selenium, Monkey Talk, Robotium and Appium. They compared these tools with respect to supported platforms, being open source, being cross platform and supported browsers. They did not include the tools' ability to test connections, gestures, memory and battery leakage or record and replay property.

Even there are a number of related studies[6][3][7], the criteria that they compared the tools with respect to, are very high level and not including detailed technical information. It is easy to reach the information they provided, by using official Web pages of the tools. On the other hand, in our study, we compared the tools with respect to 28 criteria including functional properties, connectivity properties, interruption handling capabilities, properties about test environment, memory and battery leakage handling capabilities. We have also considered documentation and community of the tools, ease of environment setup and scripting languages. We have provided detailed technical information and mentioned functional capabilities such as interaction with soft keyboard or notifications and generating test report files

which may be needed in the later stages of testing process. In addition, the criteria that we identified may be used for later comparative studies such as comparison of tools for IOS platform or the tools which will be released later.

We have conducted a case study in which we have experienced capabilities and limitations of tools. We have developed test suites using equivalence class partitioning, for two applications. We implemented these test suites using 3 tools and compared the tools. We concluded that, Robotium provides 52% of the criteria totally, Appium provides 18% of the criteria totally and UIAutomator provides 13% of the criteria in total. We have found out that there is not a single tool that meets all of the criteria.

This thesis is structured as follows:

- In Chapter 2, we give information about background technology; Android Application Development, Android Application Testing and selected frameworks; Robotium, Appium and UI Automator. We also give information about the literature review that we have conducted at different stages of our study.
- In Chapter 3, we present our research methodology in a detailed manner.
- In Chapter 4, we explain our experimental study in details.
- In Chapter 5, we present result of our study and future directions.

CHAPTER II

2. BACKGROUND TECHNOLOGY AND LITERATURE REVIEW

2.1 BACKGROUND TECHNOLOGY

In this chapter, we present general information about related technologies.

2.1.1 ANDROID APPLICATION DEVELOPMENT

Android is a mobile operating system which provides large set of features that supports mobile applications [8]. Android application development refers to the process of creating Android applications using Java.

Android applications have four types of components [9]:

- Activities: An Android Application consists of a number of activities. Activities are screens that users interact in order to perform an action such as take a photo or send an e-mail. Generally there is a main activity which is presented to the user when the application starts. In order to perform multiple actions activities call each other.
- Services: A service runs in the background to perform long running or remote processes and does not provide a user interface. For example a service enables the music to play in the background without interrupting a running application.
- Content providers: Content providers enable for query and modify data stored in file system, database or any other storage location.
- Broadcast receivers: Broadcast receivers deal with system or application broadcast announcements such as low battery, screen turn off. They do not provide a user interface. They may send notifications to alert user about the announcement.

The application development environment includes Java Development Kit, Android SDK, an IDE (Eclipse or Android Studio) and a virtual device (emulator). Android SDK provides a variety of tools that helps for developing Android applications. Virtual device which is provided by Android SDK, helps developers to develop, run and test applications without using a physical device [10].

When an Android code is compiled, an APK (Android package) file is generated. This APK file contains all the contents of the application. In order to install an Android application to the device, this APK file is used.

2.1.2 ANDROID APPLICATION TESTING

Testing is an integral part of software development lifecycle. It is a critical process as it helps "improve the quality of your apps, ensure better user satisfaction, and reduce overall development time spent on fixing defects"[11]. Thus, new testing approaches, technologies and strategies are being developed to make this process faster, much efficient and more reliable.

As mobile applications have become an essential part of our lives, importance of mobile application testing increases continuously. Mobile applications have special characteristics that make mobile application testing different from traditional and Web application testing. These typical characteristics can be listed as follows [12]:

- Mobile connectivity: Mobile applications connect to mobile networks which may be different in terms of speed, security and reliability. This property arises the need for extra functional testing performed under different connectivity scenarios.

- Limited resources: Mobile devices are far away from computers in terms of hardware resources such as RAM, disk space and CPU. Thus, resource shortage should be considered during testing process.

- Autonomy: The functionality of traditional computers relies on electricity supply. On the other hand each mobile application may require different energy consumption. For instance an application that requires continuous 3G connectivity, strongly affects the autonomy of the device. That is why; energy consumption of mobile device should be evaluated during testing process.

- New user interface: Mobile devices vary in terms of UI properties such as screen size and resolution. Mobile applications may look and differently on different UI. Thus, during GUI testing mobile applications it is necessary to consider that different mobile devices may react differently to the same application because of the differences of user interface.

- Context awareness: Mobile apps may sense the context and act according to the different contextual input such as temperature, location and brightness. The number of contextual input may be huge. Thus, context specific testing techniques and coverage criteria should be used for mobile app testing.

- Adaptation: Mobile application may adapt to the contextual information during its execution. This adaptation should be considered during testing process.

- New programming language: New frameworks, APIs, libraries and programming languages such as Objective-C is used for mobile applications. Traditional testing techniques are needed to be revised according to them.

- New Mobile Operating System: Mobile operating systems (Android and iOS) are different from computer operating systems and from each other. Moreover, new versions of mobile operating systems are released continuously. Testing approaches that may detect bugs related to the unreliability and variety of operating systems should be used for mobile application testing.

- Diversity of phones and phone makers: There are a big number of different mobile devices and vendors. It is stated that 1.800 hardware/OS different configurations exist. This situation arises the need for testing techniques to cover maximum diversity.

- Touch screens: The main input source of mobile apps are touch screens. Thus, considering touch screen functionalities is an important step of mobile app testing.

Android application testing refers to the set of activities to evaluate an Android application. As Android is a mobile operating system, Android applications have the properties stated above and these properties should be considered during testing process. Android allows running test suites on either a real Android device or a virtual Android device.

Android development kit provides 'Testing Support Library' which includes APIs to that allows developing and running test code for Android apps. Support Library is available inside Android SDK Manager. Support library also includes test automation tools; AndroidJUnitRunner, Espresso and UIAutomator [13]. Android JunitRunner is a test runner that runs Junit 3 and Junit 4 tests on Android applications [14]. Espresso and UIAutomator are UI testing frameworks. Espresso runs on single application where UIAutomator can run on any application installed on the target device.

2.1.3 ROBOTIUM

Robotium is a widely used open source Android test automation framework. It was developed by Renas Reda and first released in 2010 [15]. Robotium is mostly used for functional user interface testing, however it can be used for system and acceptance tests as well. It supports both native and hybrid application tests. Native applications are the ones which are developed for a specific platform and can be downloaded from mobile application stores. On the other hand hybrid applications are combinations of native and web applications. The difference of hybrid

applications from the native ones are that they require HTML to be rendered in a browser. Similar to native apps, hybrid apps can be installed from app store as well. [16]

When we searched on tutorials, blogs and forums, we have seen that, each of the 5 sources that are mentioned in section 3.1, states Robotium as one of the most popular open source Android testing frameworks. Its similarity to Selenium [35] which is one of the widely used test automation framework for Web applications is one of the reasons for this popularity.

The source code of Robotium is accessible on GitHub[17] and anyone can contribute to the framework's development.

2.1.4 APPIUM

Appium is a popular open source testing framework that can be used for both Android and iOS applications. The main purpose of Appium is writing one test code and running it on both platforms without changing the code. Its first release on GitHub was on 2012 by Don Cuellar. Afterwards, the company, Sauce Labs created a team for supporting Appium. In May,2014 Appium version 1.0 was released [18]. It supports native, hybrid and mobile web application tests. Mobile web applications are the ones which do not require installation and who needs to be tested using different mobile browsers [7].

As a result of our searches on most used Android testing frameworks, we have found that each of our sources mentions Appium. Being a cross-platform framework is makes Appium to go a step forward.

The source code of Appium is accessible on GitHub [19] for contribution of developers.

2.1.5 UI AUTOMATOR

UIAutomator is an automated mobile testing tool which is included in Android Testing Support Library. Unlike Robotium and Appium, UIAutomator comes with Android SDK. It requires Android 4.3 (API Level 18 or higher) [13]. UIAutomator provides APIs for building UI tests, user interactions and system events.

UIAutomator has a viewer named UIAutomatorViewer. This viewer analyzes the user interface components which are currently displayed on an Android device. It provides information about layout hierarchy and user interface components' properties such as ID, text, class. It is a viewer which is widely used. Even other mobile test automation tools get help from UIAutomatorViewer for inspecting UI components.

When we searched about the most popular mobile application testing frameworks, we have found out that 4 of 5 sources that we searched states UIAutomator as one of the most popular Android test automation tools. Its successful viewer and being included in Android SDK increases the tool's popularity. In addition, Appium uses UIAutomator internally [20].

2.2 LITERATURE REVIEW

In this part, we presented our literature review under two sections: (1) studies about mobile testing, (2) comparative studies about software development and testing.

2.2.1 MOBILE TESTING

Mobile testing is an area which has been popular recently. As mobile applications have become a major area in IT sector, mobile testing have started to get attention of researchers.

Muccini, Francesco and Esposito [12] conducted a study about whether mobile applications need any specific approach for their verification and validation or not. They answered three questions in their study. (1) Are mobile applications so different than traditional software to require special testing approach? (2) What are the new testing challenges come with mobile applications? (3) What is the role of automation in mobile testing? They concluded that; (1) Mobile applications are so different than traditional software and need specific testing approaches. (2) The new testing challenges are mostly caused by the mobility and contextual nature of mobile applications. (3) In order to keep cost of mobile applications low and detecting bugs at different layers such as operating system, application, application framework and hardware layers, automation is needed. However more detailed studies are needed for a more accurate answer.

Bayley, Flood, Harrison and Martin [21] proposed a cross platform mobile testing automation tool named MobiTest. They aimed to generate test suite running on different platforms for the same application by this tool. Their focus was GUI testing. They emphasized that, the main challenge of developing a cross platform testing tool is platform specific features. The architecture of MobiTest solved some of the issues however they stated that, there are more issues such as determining components and attributes for different platforms and gathering knowledge about how existing multi-platform applications identify components, which should be studied and solved later on. The design of the tool has not been completed yet. As future work, they aim to complete the design of MobiTest and implement the tool firstly for Android then for multiple platforms.

Shah,G , Shah,P and Muchhala[22] discussed Appium in their study. They examined the architecture of Appium, its components and how to work with Appium. As it is a cross platform tool, working with Appium is studied for iOS and Android separately. They concluded their study with advantages and disadvantages of Appium. They indicated that; mainly, being cross platform, open source, supporting multiple languages are advantages of the tool. They stated disadvantages of Appium as its technical limitation on running tests on different iOS devices and APIs older that API 16 being not supported by Appium.

2.2.2 COMPERATIVE STUDIES

There are a number of comparative study which inspired us for conducting such kind of a study.

Jain,A, Jain,M and Dhankar [23] made a comparison between software test automation frameworks, QTP and Ranorex. These tools are used for testing Web and desktop applications The authors emphasized the importance of test automation and advantages over manual testing. According to their study, automated testing is faster, needs less investment on human resources, more reliable, enables test run multiple times, helps for performing compatibility testing and more economic in long term. They gave general information about the benefits and disadvantages of the tools QTP (Quick Test Professional) and Ranorex. For comparison, they gave points between 1-10 to each of the tool according to a set of criteria. Their criteria is; cost, environment, supported browser, online support, supported coding languages, ease of learning, strongly typed, evolution with age, training cost and integration with quality assurance tools. Technical detail information was not used for comparing the tools. They concluded that, Ranorex provides significant cost benefits.

Kaur and Gupta [24] conducted a comparative study on test automation tools; Selenium, Quick Test Professional and Testcomplete which are used for Web and Windows application testingThey aimed to compare them to discover their usability and efficiency. They evaluated these tools according to the features; Licensing cost, application support, object oriented language support and scalability, support for operating system and platforms, programming skills, usage, working with database applications, platform dependency and report generation. As a result of this evaluation study, they stated that, Quick Test Professional which is the most expensive one is the best among the three.

Dalmasso,Datta, Bonnet and Nikaein [25] conducted a comparative study on cross platform mobile application development tools. They stated that, developing applications for different platforms individually requires more knowledge and effort. Cross platform development tools reduces cost and time of application development. They aimed to compare the performance of cross platform application development tools; PhoneGap, Sencha, JQueryMobile and Titanium. They developed Android

applications using each of these tools and observed their memory usage, CPU usage and power consumption.

Rani, Suri and Khatri [26] conducted a comparative study on mutation testing tools. In mutation testing, the program is modified by some small changes to model low level defects. Some of these defects are detected in testing process while some of them are not recognized as defect. The idea of mutation test is to evaluate the quality of the tests by observing the amount of detected mutations. In the paper, it is stated that manual mutation testing requires a lot of time and effort. That is why there are a number of automation tools for mutation testing. In the study, 5 mutation testing tools; MuClipse, Judy, Jumble, Jester and PIT is compared. They conducted an experimental study. They generated mutants for some programs using each tool. As a result of this experimental study, they compared the tools according to their fault finding abilities. As a result of this study, they concluded that each tool has some special features and the most suitable tool changes according to the requirements.

Kumar and Singh [27] conducted a comparative study on Web service testing tools. Web services are software components that can be accessed by different programming interfaces. They emphasized that, there are several number of Web service test automation tools. The most important criteria for these tools are performance. They selected 6 web service testing tool to compare. These tools are; Apache Jmeter, Soapui Pro, Wcf Storm, Wizdl, SOA Cleaner and SOAPSonar Personal. At first, they analyzed these tools on the basis of application support, programming language and framework, OS support, license, developer and Website. Then, they created and run test cases against temperature conversion web service using each of the tools and compared the tools in terms their performance. They compared the tools according their response time to valid and invalid inputs. They concluded that Wcf Storm has the maximum response time. Apache Jmeter and Wizdl has similar response time which is smaller than Wcf Storm. SOAPSonar Personal has the lowest response time which means it is the fastest tool among six.

CHAPTER III

3 METHODOLOGY

In this chapter we explain the methodology used in our study in a detailed manner. Our research methodology is presented in Figure 3.1. In this figure, it is shown that, we have started our study with literature review. Then we have selected the Android test automation tools to evaluate. Afterwards, we have selected Android applications to be used during different phases of our study. Then we have examined blogs, checklists and related works to identify our comparison criteria. After examination, we have written test code for a selected application to find new criteria After the criteria identification is completed, we have examined tutorials, blogs, books related studies and forums to evaluate the selected tools with respect to the identified set of criteria. Afterwards, we have conducted an experimental study in which we have developed test suites for 2 selected applications using testing tools we evaluated. Lastly, we have analyzed the results we gathered as a result of examination of resources and our experimental study.

In this chapter, in the first section, we explain the process of selecting frameworks to compare. In the second section, we give information about the process of selecting applications to test. In the third section, we presented information about the process of criteria identification. In the fourth section, we give information about writing test code using a selected framework in order to find out new criteria. In the fifth section, we presented information about the process of evaluating frameworks with respect to the criteria. Lastly, in the sixth section we give information about our case study.

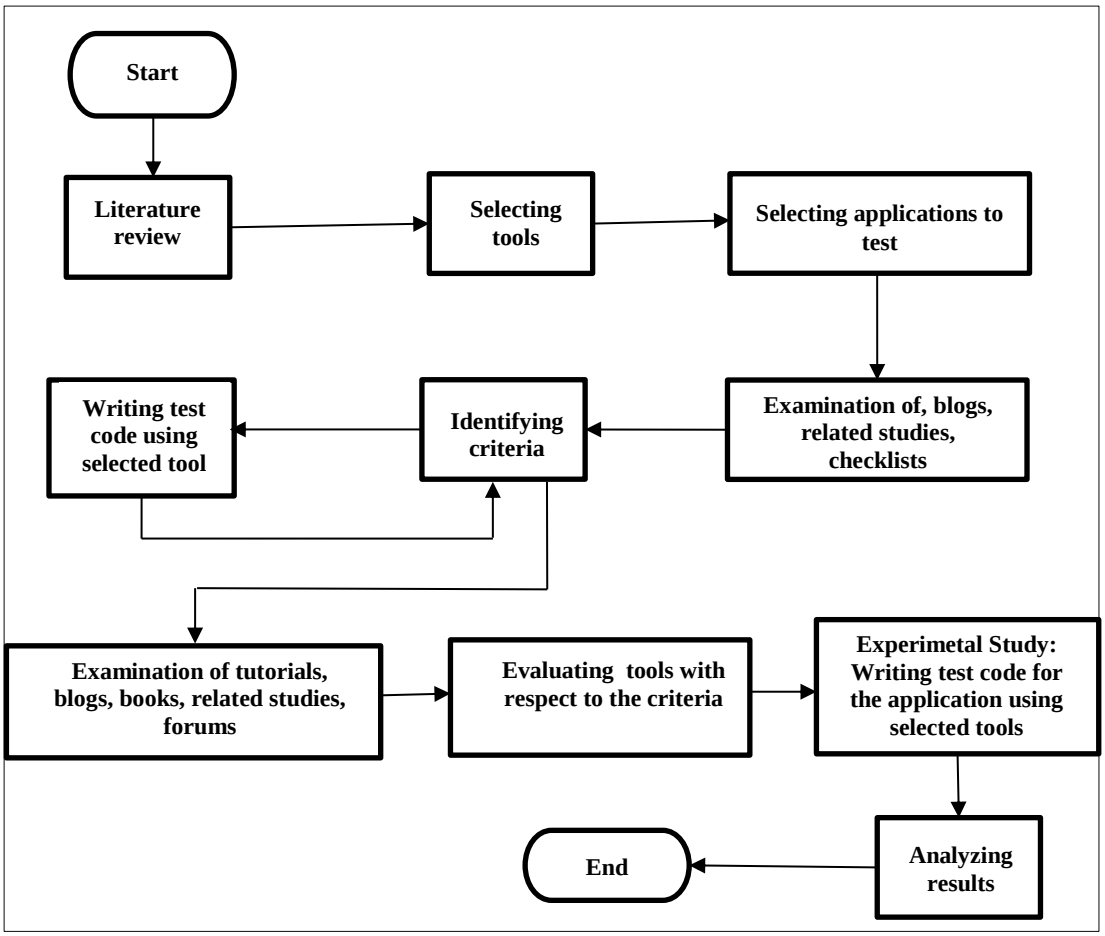

Figure 3.1: Research Methodology

3.1 SELECTING TOOLS

Mobile application testing frameworks are aimed to automatize testing process of mobile applications. There are a number of mobile testing frameworks which enables testers to conduct automatized functional, performance, acceptance and integration tests of mobile applications. In order to select tools to compare, we searched most widely used open source tools from a number of web pages, blogs and tutorials. We have selected Android as our target because of its prevalence in the market. Thus, we choose open source tools which are applicable for Android.

Sauce Labs is an American web and mobile app test automation company. “In 2015, the company was named by the San Francisco Business Times as one of the top 100 fastest growing private companies in the Bay Area for a second consecutive year, reporting 3-year revenue growth of 472%” [28]. The company listed most popular Android test automation tools as [29]:

- Calabash
- MonkeyTalk
- Robotium
- UIAutomator
- Selendroid

Testdroid is a set of products related to software development and testing by Bitbar Technologies Limited. Bitbar is a company founded in 2009 and has customers included Facebook, LinkedIn, Flipboard, Pinterest, and eBay [30]. Testdroid has a blog which includes articles about mobile application testing. This blog lists top 5 Android test frameworks as follows[31]:

- Robotium
- UIAutomator
- Espresso
- Calabash
- Appium

Optimus Information is a company headquartered in Canada. The company provides outsourced technology services to companies. Their services are about software development, mobile development, software testing and business intelligence. In their blog [36] they listed 10 most popular mobile testing tools as follows:

- Appium for Android and iOS
- Calabash for Android and iOS
- MonkeyTalk for Android and iOS
- Robotium for Android
- Selendroid for Android
- UIAutomator for Android
- UIAutomation for iOS
- Frank for iOS
- KIF (Keep It Functional) for iOS
- iOS Driver for iOS

TestingExcellence is a website founded in 2007 and provides software testing articles, tutorials, information about testing tools, conferences and news. Amir Ghahrai who is the founder of the website states the popular open source mobile automation tools as follows [37]:

- Appium for Android and iOS
- Calabash for Android and iOS
- Frank for iOS
- MonkeyTalk for Android and iOS

- UIAutomation for iOS
- Robotium for Android
- iOS Driver for iOS
- UIAutomator for Android
- KIF for iOS
- Selendroid for Android
- EarlGrey for iOS

TestLab4apps is a company interested in mobile application testing. They use test methods which combine manual and automated testing. They conduct different kinds of mobile application tests such as functional, compatibility, stress, acceptance and user interface. In their blog [38], they listed 5 open source Android app test automation tools as follows:

- Appium
- Robotium
- MonkeyTalk
- MonkeyRunner
- Sikuli

As a result of our research we have found out that, Robotium and Appium are the most widely used mobile testing frameworks. Each of the five resources that we have searched mentioned Robotium and four of them mentioned Appium. Thus, we have selected these two frameworks for our study.

UIAutomator is another widely used framework. We have found out that, four of the five resources we searched mentioned about this tool. UIAutomator differentiate from others as it is one of the test automation tools which is included in Android Testing Support Library. Android Support Library is available with Android SDK. Other tools included in Support Library are AndroidJUnitRunner and Espresso [13]. However, they are not as popular as UIAutomator. Another property of UIAutomator is that, it provides API for other tools. For instance Appium uses UIAutomator driver.

Because of these distinguishing properties and its popularity, we have selected UIAutomator for our study too.

As a result, Appium, Robotium and UIAutomator are selected to be compared in our study.

3.2 SELECTING APPLICATIONS TO TEST

One small sized and two medium sized open source applications are selected to be used in different phases of our study.

For identifying criteria, a small sized application; Android Scientific Calculator is selected. It is a scientific calculator whose GUI is shown in Figure 3.2.1.

Figure 3.2.1: Calculator Application UI [1]

We have downloaded the source code of the application[39] and wrote test cases based on equivalent class partitioning for this application. During this process we have determined some criteria that were not mentioned in checklists that we have studied.

We have conducted a case study as well. For this case study we have selected two open source medium sized applications, DroidWeight and 2048. Droidweight is a health application that aims to track weight and some other body parameters. It is an application that can be downloaded from Google Play Store [40]. Three main screens of the application is shown in Figure 3.2.2.We have downloaded the source code of the application from GitHub [41].

[1] Code Monkey. (2013). Calculator (open source) (Version 2.0.1) [Mobile Application Software]. Retrieved from https://sourceforge.net/projects/androidcalculat/

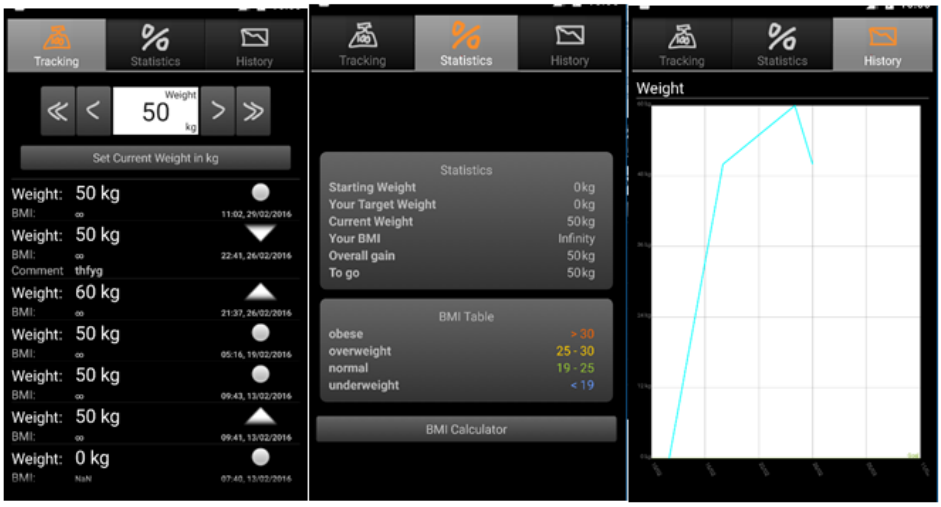

Figure 3.2.2: Droidweight application UI[2]

2048 is a popular mobile game that is available on both Android and iOS platforms. There are many different versions of the application on Google Play Store. For our study, we have downloaded a version which enables users to play offline [42]. 2048 is a hybrid application that includes a webview. This property allows us to determine the capabilities of the testing frameworks on testing hybrid applications. The main screen of the application is shown in Figure 3.2.3.

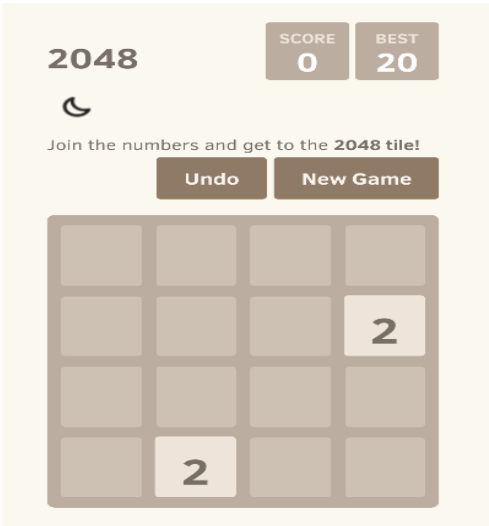

Figure 3.2.3: 2048 Application UI [3]

[2] Aymanstar.(2013). droidweight (Version 17) [Mobile Application Software]. Gathered from https://github.com/aymanstar/droidweight
[3] uberspot. (2016). 2048 (Version 2.06) [Android Application Software]. Gathered from https://github.com/uberspot/2048-android

Table 3.2.1 shows the properties of three Android applications, Calculator, Droidweight and 2048.

Table 3.2.1: PROPERTIES OF ANDROID APPS USED IN OUR STUDY

	Calculator	Droidweight	2048
Line of code (LOC):	835	3966	168
Number of packages:	2	10	2
Number of classes	10	48	12
Number of methods	31	295	8
Number of screens	1	3	1

DroidWeight and 2048 are open source applications whose source code is not difficult to read and to understand. This property simplifies the process of understanding the applications and developing test cases. Moreover, these applications provide functionalities that enable us to test capabilities of the frameworks. For instance, drag and drop property of the test tool can be tested on 2048 effectively. Because of these reasons we have selected these two applications for our case study.

3.3 IDENTIFIYING CRITERIA

In order to identify criteria to evaluate testing frameworks, firstly, we have searched mobile testing checklists. We have selected the criteria related to GUI testing, interrupt testing and memory/storage leakage testing.

Test management approach (TMap) which is a software testing organization, combines insights and techniques about testing [32]. TMap released a checklist [33] for mobile application testing. Characteristics, which is needed to be tested are listed under five parts;

- Device Specific Checks present characteristics which are related to the device that the application is installed. There are 27 device specific checks in TMap's checklist.
- Network Specific Checks present characteristics which are related to network connection. There are 9 network specific checks in TMap's checklist.
- App Specific Checks present frequently used functionalities of the application. There are 14 app specific checks in TMap's checklist.

- App User Interface Checks present characteristics that provide a better user experience. There are 19 app user interface checks in TMap's checklist.
- Store Specific Checks, which are gathered with the help of Apple store guidelines, present characteristics needed to be tested before submitting the application to the store. There are 35 store specific checks in TMap's checklist.

As we are interested in GUI based functional testing, interrupt testing and memory and battery leakage testing, some of the characteristics, mostly under the parts 'App User Interface Checks' and 'Store Specific Checks', listed in the TMap's checklist were out of our scope. App user interface checks are related to user experience rather than the functionality of the app. Store specific checks are not necessary unless the application will submitted to the store. Other characteristics inspired us for selecting criteria in our study.

RapidValue Solutions[34] published an article that provides a checklist for selecting suitable mobile test automation tool. The checklist includes requirements that mobile application testing tools must have and the tools that meets these requirements. These requirements are listed under following parts;

- Identification methods: Capability of the tool for identifying objects.
- Devices exactly like the actual user has: No Jailbreak/Rooting: The tool's not requiring any changes in the device under test.
- Supported phones
- Plugs-in to existing test environments: Ability of the tool to integrate into testing environment.
- Recorder: Having capability to record and replay tests.
- Same test can run on different devices
- Test Execution: The speed of execution and the tool's ability to run serial tests.
- Full Functionality Support: Functionalities that the tool should be support such as gestures.
- Device connectivity: The tool's ability to test connectivity of the device

Another article is"Testing Checklist for Mobile Applications" by Anurag Khode [35]. In this article, 26 test cases and expected results of them was listed. These are the tests which should be run when a mobile application is tested. The test cased are mainly about interrupt testing, such as call/sms handling, removal of battery/charger, etc. Tests to provide a better user interface such as installation, application logo, exit application were also stated in the article. However there were not any test case related to GUI testing.

In addition to the checklists, we have examined blogs [43][44] and the related studies [3][6][7] discussed in Chapter 2.

To determine the criteria list, we have searched about the behaviors, properties, requirements and possible user actions of mobile applications that are needed to be tested. We have gathered about 100 criteria. The criteria related to installation, integration, usability and store (such as Google Play and Apple Store) are out of our scope, thus we have not included these criteria in our study. We have included the criteria related to GUI, network, interruption and memory/battery leakage as they are our main focus. Moreover, we have also added some criteria which give general information about tools such as test language, documentation and community of the tool, ease of setup. As a result of this research we have defined 19 criteria to be used in our study.

We have also conducted an exploratory study to identify additional criteria. We have created test suite for Scientific Calculator App and have developed test code using Robotium, Appium and UIAutomator. During this process we have found 9 new properties needed to be tested using automation tools and not stated in the resources we searched. Thus we have also added these properties to our criteria.

As a result of our search and exploratory study we have identified 28 criteria which we have compared our selected tools with respect to. These criteria can be used for evaluation of other mobile testing tools as well. Table 3.3.1 shows resulting criteria.

Table 3.3.1: EVALUATION CRITERIA FOR MOBILE TEST AUTOMATION TOOLS

	Criteria
C1	Can we use delays in test code
C2	Can we test gestures
C3	Can we select different options at the same time
C4	Can the tool take screenshot
C5	Is it possible to reach "back", "recent apps" and "home" buttons using the tool
C6	Is interaction with soft keyboard available
C7	Can we test the behavior of the app with different internet connection (no connection, Wi-Fi, 3G, 2G)
C8	Can we test behavior of the app on airplane mode
C9	Is interaction with status bar notifications available
C10	Is it possible to generate test report file using the tool
C11	Does the tool have record and replay property
C12	Can we test the behavior of the app when a there is an incoming call/sms
C13	Can we test the behavior of the app when a there is a popup alert (alarm, calendar)
C14	Can we test the behavior of the app when a there is an incoming push message from another app
C15	Can we test if inserting and removing charger causes any problem or not

C16	Can we test the behavior of the app when the battery is low
C17	Can we test the behavior of the app when the memory is low
C18	Can we test on emulator
C19	Can we test on real device
C20	Is testing with APK file available
C21	Is testing with source code available
C22	Test language
C23	Can we execute tests using command prompt
C24	Can we test hybrid applications
C25	Is it possible to test an application without developer level knowledge about the application code
C26	Is it fast & easy to setup working environment
C27	Is the documentation about the tool enough
C28	Is the community active

- Criteria 1 represents whether the tool enables inserting waits inside the test code for different purposes such as waiting for a new window to open.
- Criteria 2 represents the ability of the tool to simulate different user gestures. The gestures we investigated are given in Table 3.3.2.
- Criteria 3 represents the ability of the tool to simulate selecting more than one option at the same time such as clicking on different points on the screen at the same time. Multiselection refers to the user action in which the user clicks on more than one element on the screen simultaneously. Such interaction could be an undesired one or unintentionally performed. In mobile application testing process it is needed to test the behavior of the application when this situation occurs. Thus, we wanted to represent the tool's ability to simulate multiselection with this criteria.
- Criteria 4 represents the ability of the tool to take screenshot of the virtual or real mobile device.
- Criteria 5 represents the ability of the tool to reach three main Android buttons; back, recent apps and home
- Criteria 6 represents the ability of the tool to interact with soft keyboard, such as clicking on soft keyboard buttons and closing soft keyboard.
- Criteria 7 represents the ability of the tool to change internet connection. The options are turning on and off WI-FI, turning off mobile data transfer, opening 3G and 4G mobile data transfer
- Criteria 8 represents the ability to of the tool to change the device mode to Airplane mode
- Criteria 9 represents the ability of the tool to reach and interact with the notification area.
- Criteria 10 represents the ability of the tool to generate and export test report as a file.

- Criteria 11 represents the ability of the tool for recording tests from emulator or real device and play back the recorded tests.
- Criteria 12 represents the ability of the tool to simulate an incoming call and text message.
- Criteria 13 represents the ability of the tool to simulate popup alerts such as alarm and calendar alerts.
- Criteria 14 represents the ability of the tool to simulate push messages from another applications.
- Criteria 15 represents the ability of the tool to simulate inserting and removing charger to the device.
- Criteria 16 represents the ability of the tool to change battery level of the device.
- Criteria 17 represents the ability of the tool to change used memory level of the device.
- Criteria 18 represents the ability of the tool to run test cases on emulator (virtual device).
- Criteria 19 represents the ability of the tool to run test cases on a real physical device.
- Criteria 20 represents the ability of the tool to create test suite and run test code using APK file of the application.
- Criteria 21 represents the ability of the tool to create test suite and run test code using source code of the application.
- Criteria 22 represents the language used for writing test code using the tool
- Criteria 23 represents the ability to run tests created by the tool using command prompt.
- Criteria 24 represents the ability of the tool to create and run tests for hybrid applications. Hybrid applications are apps which are combination of web and native applications. They are developed for a specific platform and run inside a native container like native mobile apps. However, they also include webviews that are built with web technologies such as HTML, CSS and JavaScript [45].
- Criteria 25 represents the ability to create and run tests using the tool, without deep knowledge about the source code of the application such as name and functionalities of the methods, classes and packages and id of UI elements
- Criteria 26 represents the ease of preparing working environment of the tool such as downloading and installing necessary software, importing libraries and jar files.
- Criteria 27 represents if available documentation (tutorials, blogs, videos, books) provides sufficient information to use the tool efficiently or not.
- Criteria 28 represents if the community of the tool such as Google Groups, Github page and forums are active or not.

As criteria 2; 'Testing Gestures' consists of different gestures, we have also created a table that shows each gesture separately. Table 3.3.2 shows these gestures.

Table 3.3.2: GESTURES FOR EVALUATING MOBILE TEST AUTOMATION TOOLS

Gesture	Definition
Drag	The gesture consists of clicking on an object, moving it across the screen and releasing it.
Pinch	Two fingers are put on the screen and the gap between fingers shrinks or expands. When this gap shrinks, the gesture is named as pinch-in and when the gap expands the gesture is named as pinch-out. This gesture is mainly used for zooming.
Swipe	Similar to drag gesture, swipe gesture consists of the activities tapping on the screen with one finger, move the finger across the screen and release. The main difference between drag and swipe is that, swipe gesture does not have a target screen element while drag has [46].
Long Click (Long Press)	Long click (long press) means clicking on the screen with one finger, waiting for a while in soconds and release.

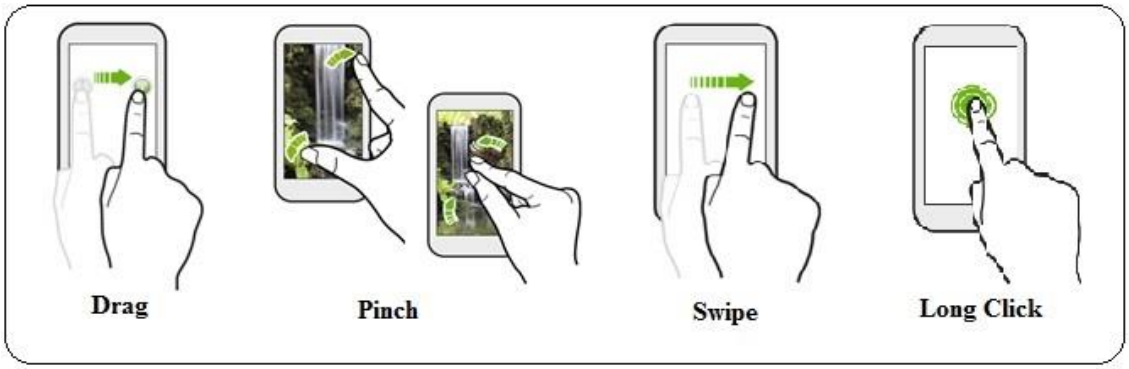

Figure 3.3.1 : Gestures[4]

3.4 EVALUATING THE TOOLS WITH RESPECT TO THE CRITERIA

After criteria identification, we have evaluated mobile applications testing tools with respect to our criteria set. For each of the criteria, we have searched on papers, tutorials, blogs, books and related studies to determine whether the tool meets the criteria or not. If the information we needed was not stated any of the sources, we

[4] **Adapted from** http://www.htc.com/us/support/htc-one/howto/cat_54453.html

have opened issues on GitHub, Google Groups or forums such as Stack Overflow which is a web site that includes questions and answers related to programming.

3.4.1 ROBOTIUM

We have evaluated Robotium according to the criteria represented in Table 3.3.1. In this part, we explain the ability of Robotium to meet each of the criteria.

C.1 Insert Delays into Test Code

Robotium allows simulating delays by inserting waits inside the code. These delays may be used for either simulating a user wait or system wait such as waiting for a new window to open.

In order to test delays, Robotium provides the method, solo.sleep(time) in which time refers to the miliseconds that we wanted to wait for.

C.2 Testing Gestures

Robotium allows for simulating mostly used gestures [47].

- In order to simulate drag gesture it is possible to use the method;

```
solo.drag(fromX, toX, fromY, toY, stepCount)
```

The parameters ‘fromX’ and ‘fromY’ refer to the coordinates of the initial point where drag action starts.

The parameters ‘toX’ and ‘toY’ refer to the coordinates of the last point where drag action ends.

The parameter stepCount refers to the number of steps to complete drag action.

- In order to simulate swipe gesture it is possible to use the method;

```
solo.swipe(startPoint1, startPoint2, endPoint1, endPoint2);
```

Robotium allows swipe gesture with two fingers. The parameters ‘startPoint1’ and ‘startPoint2’ refer to the initial points of the first and second fingers. The parameters ‘endPoint1’ and ‘endPoint2’ are the end points for two fingers .

- In order to simulate pinch gesture, it is possible to use the method;

```
solo.pinchToZoom(startPoint1, startPoint2, endPoint1, endPoint2);
```

The parameters ‘startPoint1’ and startPoint2 refer to the initial points of two fingers. The parameters ‘endPoint1’and ‘endPoint2’ refer to the last points of two fingers.

- In order to simulate long click gesture, Robotium provides a number of different methods. Two of these methods are;

    - `solo.clickLongOnText(text);`

      Using this method it is possible to long click on the screen where the parameter ‘text’ appears.

    - `solo.clickLongOnScreen(x, y);`

      Using this method, it is possible to long click on a point whose coordinates are stated as ‘x’ and ‘y’.

C.3 Selecting Different Options At The Same Time

Robotium resources do not provide information whether selecting different options at the same time is possible or not using the tool. We have examined the API too, but could not find any method or functionality to simulate multi selection in Robotium API

C.4 Taking Screenshot

Robotium allows taking screenshot of the real or virtual device using the method;

```
solo.takeScreenshot();
```

All screenshots are stored on the folder /sdcard/Robotium-Screenshots/. In order to save the screenshot it is necessary to give permission to main application to write SD Card. For this purpose, it is necessary to add following line of code to the manifest file of the application under test [48].

```
   <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE">
```

C.5 Reaching "Back", "Recent Apps" And "Home" Buttons

After our research, we have concluded that Android hardware buttons; back, search, home and recent apps can be reached by Robotium using `solo.sendKey()` method. However, with our case studies and a deeper research, we found that clicking on Android hardware buttons is not working properly. The tool API contains the static values KEYCODE_HOME and KEYCODE_APP_SWITCH to be used as `solo.sendKey(KeyEvent.KEYCODE_HOME)` and `solo.sendKey(KeyEvent.KEYCODE_APP_SWITCH)` but these methods are not

functional and do not trigger any action.Only “Back” button is functional by `solo.sendKey(KeyEvent.KEYCODE_BACK)` or `solo.goBack()` method.

C.6 Interaction With Soft Keyboard

Pressing on soft keyboard generates key events. These key events are represented by key codes [49].

Robotium allows interaction with soft keyboard by sending key code with the command;

```
solo.sendKey(key code);
```

Hiding the soft keyboard is also possible using the method;

```
solo.HideSoftKeyboard();
```

C.7 Different Internet Connection (No Connection, Wi-Fi, 3G, 2G)

Robotium allows users to set WI-FI and mobile data as turned on or turned off. However, it is not possible to set network type as 2G, 3G or 4G. Methods for these settings are [50] ;

```
solo.setMobileData(turnedOn);
solo.setWiFiData(turnedOn);
```

These methods gets boolean parameters (true or false).

Changing those states needs permissons in the AndroidManifest.xml file of the application under test. It is necessary to add permissions for changing network state and changing network state of the application using following lines of code:

```
<uses-permission android:name="
android.permission.CHANGE_NETWORK_STATE">
<uses-permission android:name="
android.permission.CHANGE_WIFI_STATE ">
```

C.8 Airplane Mode

Robotium resources do not provide information if changing device mode to airplane mode is possible or not using the tool.We have also examined the API to find a method to change the device to airplane mode but could not find any related method.

C.9 Interaction With Status Bar Notifications

Interaction with status bar notification such as pulling down the notification area and click on notification is not possible using Robotium [43].

C.10 Generating Test Report File

Robotium allows users to generate test reports as XML files. After running a test, a button: 'Test Run History' appears on Eclipse. Using 'Export' option, we have downloaded our test report as XML file.

C.11 Record And Replay Property

Robotium does not have record and replay functionality itself. However there is another product; Robotium Recorder whose developers define as; "Robotium Recorder is the result of everything that we have learned from creating and continuously working with the Robotium framework for the last 4 years." [51] Robotium Recorder is available for Eclipse and Android Studio as plugin. Record and reply functionality is enabled by this tool. However, unlike Robotium, Recorder is not free.

C.12 Simulating An Incoming Call/Sms

In Google Group, the developer of Robotium Renas Reda states that, simulating an incoming call or sms is not possible using Robotium [52].

C.13 Simulating Popup Alert (Alarm, Calendar)

Robotium resources do not provide information if it is possible to simulate popup alerts using the tool or not. In addition, during test development, we could not find any Robotium method or functionality to simulate popup alerts.

C.14 Simulating An Incoming Push Message From Another App

Robotium resources do not provide information if it is possible to simulate incoming push messages using the tool or not. When we examined the Robotium API, we could not find any method to simulate an incoming push message either.

C.15 Simulating Inserting And Removing Charger

Robotium resources do not provide information if it is possible to simulate inserting and removing charger to the device or not. During test case development, we could not find any Robotium method that enables to simulate inserting or removing charger.

C.16 Testing The Behavior Of The App When The Battery Is Low

Robotium resources do not provide information if it is possible to simulate the situation that the battery of the device is low or not. When we examined API we could not find any method or function to simulate low battery either.

C.17 Testing The Behavior Of The App When The Memory Is Low

Robotium resources do not provide information if it is possible to simulate the situation that the memory of the device is low or not. When we examined API we could not find any method or function to simulate low memory either.

C.18 Testing On Emulator

It is possible to run Robotium tests on emulator. For our case study we have run all tests on emulator.

C.19 Testing On Real Device

It is clearly stated that Robotium allows users to run tests on real mobile devices [16].

C.20 Testing With APK File

Robotium allows users to test with APK file as well as with source code. Hovewer, in order to test with it, the APK file needs to have the same certificate signature with the test project. Signature matching is a process which has some compelling steps. If the certificate signature is known, it is necessary to use the same signature in test project [53]. If the certificate signature is not known, it is necessary to delete the signature and use the same signature for both the application and the test project. If the application is not signed, then it is necessary to sign the application APK. In order to sign the application, following lines of command is used:

```
jarsigner -keystore ~/.android/debug.keystore -storepass android -
keypass
android Application.apk androiddebugkey
zipalign 4 Application.apk TempApplication.apk
```

After these commands it is necessary to change the name of TempApplication.apk to Application.apk.

If the application is signed, then fistly it is necessary to delete the application sign by deleting META_INF folder of the application. Afterwards,the application is resigned using the commands presented above.

Thus, testing with APK file is possible but not as easy as testing with source code .

C.21 Testing With Source Code

Robotium allows user to test with the application's source code. In order to test with source code, the source code of the application is downloaded and a new Android test project is created. Application under test is selected as the target project of the test project. This step adds Android Manifest file of the test project the code:

```
<instrumentation android:targetPackage="package name of the app
under test" />
```

Then a new Java class is created inside the test project. It is necessary to import the activity that is aimed to test in the test activity.

In addition, it is also possible to create and run test suites for the applications that are cloned from Github by following the same steps explained above.

C.22 Test Language

The language used for coding the test cases for Robotium is Java.

C.23 Executing Tests Using Command Prompt

It is possible to execute tests using command prompt with Robotium. In order to accomplish it, firstly it is necessary to move to the directory where adb.exe file is located. This file is under android-sdk-windows\platform-tools directory.

Then, in order to execute tests, following line of command is entered on command line:

```
adb shell am instrument -w test package
name/android.test.InstrumentationTestRunner
```

C.24 Testing Hybrid Applications

Hybrid application is a combination of a web application and native Android application and includes a web view. "Web view is a control in android which is used to display web pages within itself." [54]

In the GitHub page of Robotium, it is clearly stated that Robotium has a full support for both native and hybrid Android applications [17].

In our case study, we have developed and run test suite for the application 2048 which is a hybrid application. While we could be able to reach UI elements of the native application; DroidWeight by their id, it is not possible for an hybrid application.

C.25 Need For Developer Level Knowledge About The Code

When we searched on tutorials, it is stated that; “Robotium helps us to quickly write powerful test cases with minimal knowledge of the application under test.” On the other hand, even it is possible to test with apk file, without any knowledge about source code, we have concluded that it is not an effective way of testing with Robotium because of the following reasons;

- UI elements are reachable by their index and texts. However, sometimes Robotium faces problems finding elements by these information. Thus, reaching UI elements by their ID might be more effective.

- When creating a test project using Robotium, it is needed to find package and launcher activity name of the application under test. Without knowledge about source code, it needs some complicated process to reach that information.

- Taking screenshot is possible with Robotium. However, this functionality needs permission which should be added to Android.Manifest file of the application under test. Thus, reaching and modifying the Android.Manifest file of the project is mandatory to use screenshot functionality of Robotium.

C.26 Working Environment Setup

Setting up working environment is not a complicated or time consuming process. The setup process is as follows. Using Eclipse or Android Studio, a new test project is created. Robotium jar file is downloaded and added to the test project by using “Add External Jars” menu of the IDE. After these steps which require a couple of minutes, environment setup to work with Robotium is completed.

C.27 Documentation

As Android testing is a new area, the number of documentation gathered from Internet sources, such as forums, blogs and tutorials are more informative, up to date and comprehensive than books. This documentation may be considered as satisfactory for Robotium.

C.28 Community

Robotium is one of the most popular open source Android testing tools. When we searched the mostly used Android testing tools, each source that we searched mentioned Robotium. As a result of this popularity, Robotium community is highly active.

Developer of Robotium; Renas Reda is very active on the Github page of the tool. By April,2016 there are 39 open, 696 closed issues on Robotium Github page [17].

Morever, there is a Robotium developers group in Google Groups. In this group Robotium users interact with each other and Renas Reda is also actively joins discussions [55].

On stackoverflow.com By April, 2016 there are 977 questions with the "robotium" tag [56].

These numbers may be considered as indicators and it is possible to say that the Robotium community is remarkably active.

3.4.2 APPIUM

We have evaluated Appium according to the criteria represented in Table 3.3.1 In this section, we evaluate the ability of Appium to meet each of the criteria.

C.1 Insert Delays into Test Code

It is possible to simulate delays by inserting waits inside the code using Appium. For this purpose `Thread.sleep(milliseconds)` method which simulates waits in milliseconds can be used.

C.2 Testing Gestures

Appium provides ability to simulate most used gestures; drag, swipe, pinch and long click [57].

- In order to simulate drag gesture it is possible to use following lines of code;

```
TouchAction touchAction = new TouchAction(driver);
touchAction.press(x1,y1).perform();
touchAction.moveTo(x2,y2).release().perform();
```

Firstly, we create an instance of a Touch Action class. This class provides methods to perform different mobile gestures. Then, we use press() method and state the coordinates of the point that we want to start drag action as parameters. Later, using moveTo() method we state the coordinates of the point that we want to end drag action as parameters.

- Appium allows to swipe horizontal or vertical. In order to simulate swipe gesture it is possible to use following line of code:

```
driver.swipe(startx, starty, endx, endy, duration);
```

In this method, the parameters ‘startx’ and ‘starty’ represents the coordinates of the initial point that swipe action starts. The parameters ‘endx’ and ‘endy’ represents the point where swipe action ends. The parameter ‘duration’ is the time in milliseconds to complete swipe action.

- In order to simulate pinch gesture it is possible to use the method;

```
driver.pinch(x,y);
```

In this method the parameter x and y refer to the coordinates to terminate pinch action.

In order to simulate pinch gesture, it is also possible to use MultiTouchAction class. The following lines of code shows an example of using MultiTouchAction class for pinch gesture [58] :

```
MultiTouchAction multiTouch = new MultiTouchAction(driver);
TouchAction tAction0 = new TouchAction(driver);
TouchAction tAction1 = new TouchAction(driver);

tAction0.press(scrWidth/2,scrHeight/2).waitAction(1000).moveTo
(0,60).release();
tAction1.press(scrWidth/2,scrHeight/2+40).waitAction(1000).mov
eTo(0,80).release();

multiTouch.add(tAction0).add(tAction1);
multiTouch.perform();
```

In the piece of code given above, ‘scrWidth’ refers to the width of the screen and ‘scrHeight’ represents the height of the screen. In tAction0, the finger is pressed on the center of the screen and moved to y axis. In tAction1, the finger is pressed slightly down on the center of the screen and moved to y axis.

- In order to simulate long click gesture it is possible to use following lines of code;

```
TouchAction action = new TouchAction((MobileDriver)
driver).longPress(WebElement);
action.perform();
```

In this piece of code, a touch action is created and performed. In this touch action, long click on an element on the screen (represented as WebElement) is simulated.

C.3 Selecting Different Options At The Same Time

Appium provides ability to simulate selecting different options at the same time. For this purpose, 'MultiTouchAction' class is used. Using this class it is possible to generate a set of touch actions.

```
      MultiTouchAction action = new MultiTouchAction((MobileDriver)
driver);

      TouchAction action1 = new TouchAction((MobileDriver)
driver).longPress(x1, y1);
      TouchAction action2 = new TouchAction((MobileDriver)
driver).longPress(x2, y2);

      action.add(action1).add(action2).perform();
```

In the piece of code presented above, firstly an instance is created from MultiTouchAction class. Then two long press action is defined. 'action1' is a touch action that performs long click on a point whose coordinates are 'x1' and 'y1'. 'action2' is a touch action that performs long click on another point whose coordinates are 'x2' and 'y2'. Lastly, these actions are added to multi touch action (action) and performed simultaneously.

C.4 Taking Screensot

Appium allows for taking screenshot and define the location where it is desired to store the screenshot. Using following lines of code, it is possible to take and store screenshot.

```
      String file = "file name";
      File srcFile=((TakesScreenshot)driver)
      .getScreenshotAs(OutputType.FILE);
      new File(file).mkdirs();
      String destFile = "screenShot.png"
      FileUtils.copyFile(srcFile,new File(file + '/' + destFile));
```

In the first line, the name of the file where the screenshot will be located is defined. In the second and third line, screenshot is captured. Then, the directory where the screenshot will be located is created. After that, name of the screenshot is defined. In the last line, the screenshot is stored under the defined directory and with the given name.

C.5 Reaching "Back", "RecentApps" And "Home" Buttons

Appium allows reaching and clicking on Android hardware buttons; 'Back', 'RecentApps' and 'Home'. It is possible to reach those buttons using Android key codes [48].

- `driver.pressKeyCode(AndroidKeyCode.BACK)` method is used to press 'Back' button.

- `driver.pressKeyCode(AndroidKeyCode.HOME)` method is used to press 'Home' button.

- `driver.pressKeyCode(AndroidKeyCode.KEYCODE_APP_SWITCH)` method is used to press 'RecentApps' button.

C.6 Interaction With Soft Keyboard

Interacting with soft keyboard is possible using Appium. For this interaction following methods are used;

- `driver.getKeyboard().sendKeys(String s)` allows writing string s to the text field using soft keyboard.

- `driver.pressKeyCode(int keycode)` allows clicking on soft keyboard button whose keycode is given as parameter.

- `driver.hideKeyboard()` hides the soft keyboard.

C.7 Different Internet Connection (No Connection, Wi-Fi, 3G, 2G) & C.8 Airplane Mode

Appium allows to make connection and mode settings with a single line of code presented below. However, similar to Robotium,it is possible to turn on/turn off WI-FI and mobile data but it is not possible to set network type as 2G, 3G or 4G.

```
driver.setNetworkConnection(int value);
```

The paramater 'value' is assigned according to the desired state shown in Figure 3.4.2.1.

Value (Alias)	Data	Wifi	Airplane Mode
0 (None)	0	0	0
1 (Airplane Mode)	0	0	1
2 (Wifi only)	0	1	0
4 (Data only)	1	0	0
6 (All network on)	1	1	0

Figure 3.4.2.1: Appium Connection States[5]

The figure indicates that,

- When it is needed to set mobile data, Wi-fi and airplane mode off, the parameter, value should be set to 0.
- When it is needed to set mobile data and Wi-fi off, airplane mode on, the parameter, value should be set to 1.
- When it is needed to set mobile data and airplane mode off, Wi-fi on, the parameter, value should be set to 2.
- When it is needed to set Wi-fi and airplane mode of, mobile data on the parameter, value should be set to 3.
- When it is needed to set mobile data and Wi-fi on, airplane mode off, the parameter, value should be set to 4.

C.9 Interaction With Status Bar Notifications

Appium allows opening notifications and click on them.

`driver.openNotification()` method opens the notification area. After opening the area, it is possible to click on the area using xpath, class name or coordinates of the screen element that is needed to be reached.

5 https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/network_connection.md

C.10 Generating Test Report File

Appium uses TestNG which provides test reports. TestNG is an open source testing framework similar to JUnit but includes additional functionalities [59]. It can be used not only for unit testing but also other types of tests such as integration, tests, end to end tests. After executing a test suite, a folder, named 'test-output' is created automatically. Under this folder, generated test reports with different formats are listed.

C.11 Record And Replay Property

Appium has an 'Inspector' to record and replay tests. Appium Inspector allows recording tests and replaying them when the apk file of the applications is given. Appium detects necessary information, such as package name and launcher activity name automatically after APK file is selected.

However, Inspector has two limitations [60][61][62];

- It can be used only for native apps, hybrid application cannot be tested using Appium inspector**.**
- It does not support Windows platform properly. It fails inspecting elements and record and replay function is not available on Windows.

C.12 Simulating An Incoming Call/Sms

On Appium GitHub page [63] and the blog of Rapid Value Solution [30], it is stated that simulating an incoming call or sms is not possible using Appium.

C.13 Simulating Popup Alert (Alarm, Calendar)

Appium resources do not provide information if simulating a popup alert is possible or not using the tool. When we examined API, we could not find any method to simulate a popup alert either.

On the other hand it is possible to interact with them if any. When there is an incoming alarm, using `driver.swicthTo().alert` method it is possible to reach the alert and then it is possible to click any button on the alarm popup.

C.14 Simulating An Incoming Push Message From Another App

Appium resources do not provide information if simulating an incoming push message possible or not using the tool. During test development, we could not find any function that simulates an incoming push message either.

C.15 Simulating Inserting And Removing Charger

Appium resources do not provide information if simulating inserting or removing charger is possible or not using the tool. During test development, we could not find any method or function to simulate inserting or removing charger.

C.16 Testing The Behavior Of The App When The Battery Is Low

Appium resources do not provide information if simulating low battery is possible or not using the tool. When we examined the API, we could not be able to find a method or function for simulating low battery.

C.17 Testing The Behavior Of The App When The Memory Is Low

Appium resources do not provide information if simulating low memory is possible or not using the tool. When we examined the API, we could not be able to find a method or function for simulating low memory.

C.18 Testing On Emulator

It is possible to run Appium tests on emulator. For our case study we have run all tests on emulator.

C.19 Testing On Real Device

It is clearly stated that it is possible to run Appium tests on real device [64].

C.20 Testing With Apk File

As a blackbox testing tool, Appium allows for testing with apk file. Using Appium Inspector it is possible to get the properties (such as id, index, name, class) of UI components of the target application. As Inspector does not work properly on Windows, it is also possible to use UIAutomatorviewer whose details are explained in section 3.1 to detect UI elements. Then using these information, it is possible to write test cases for the application. It is also possible to test applications with apk file using the record and replay property of Appium Inspector.

C.21 Testing With Source Code

Appium allows testing applications whose source code is available. In our case study we have developed and run test cases for the applications with source code.

C.22 Test Language

Any WebDriver compatible language such as Java, Objective-C, JavaScript,PHP, Python, Ruby, C#, Perl may be used for writing tests using Appium [65].

C.23 Executing Tests Using Command Prompt

It is possible to execute Appium tests on command prompt. However, it is necessary to use a build tool such as Ant [66] [67]. Executing TestNG tests using Ant is a painful process. Firstly, it is necessary to create a testng.xml file under project directory. This file contains information such as package name, class name and method name. Order of test cases can be also mentioned here. Then, test execution can be triggered using the command:

```
java -cp "directory name \testng.jar; directory name\jcommander.jar"
org.testng.TestNG testng.xml
```

C.24 Testing Hybrid Applications

Appium allows testing hybrid applications as well as native applications. However, Appium Inspector does not work properly with hybrid applications. The tool is not stable for recognizing webview elements. A webview element could be reached when a test suite is executed but could not be reached when the same test suite is executed later on.

C.25 Need For Developer Level Knowledge About The Code

Appium allows testing with minimal knowledge about the source code of the application. Even the information needed to test an apk file such as package name and launcher activity name can be reachable using Appium Inspector easily. Thus, testing with Appium does not need developer level knowledge about the source code of the application under test.

C.26 Working Environment Setup

The process to setup working environment for Appium needs a considerable effort. In addition to the software Eclipse, Android SDK and JDK that are needed by other tools as well, Appium requires more software. This additional software can be listed as:

- TestNG which is a testing framework that covers various testing types
- Selenium Server JAR is needed as Appium uses Selenium Web Driver for test automation

- Appium Server

After installation, it is needed to make necessary settings on Appium Server. Figure 3.4.2.2. shows the Android Settings page of Appium. On this page, Platform Name, Automation Name, Platform Version and Device Name must be set. The same settings should be made inside setUp() method of the Appium test project as well.

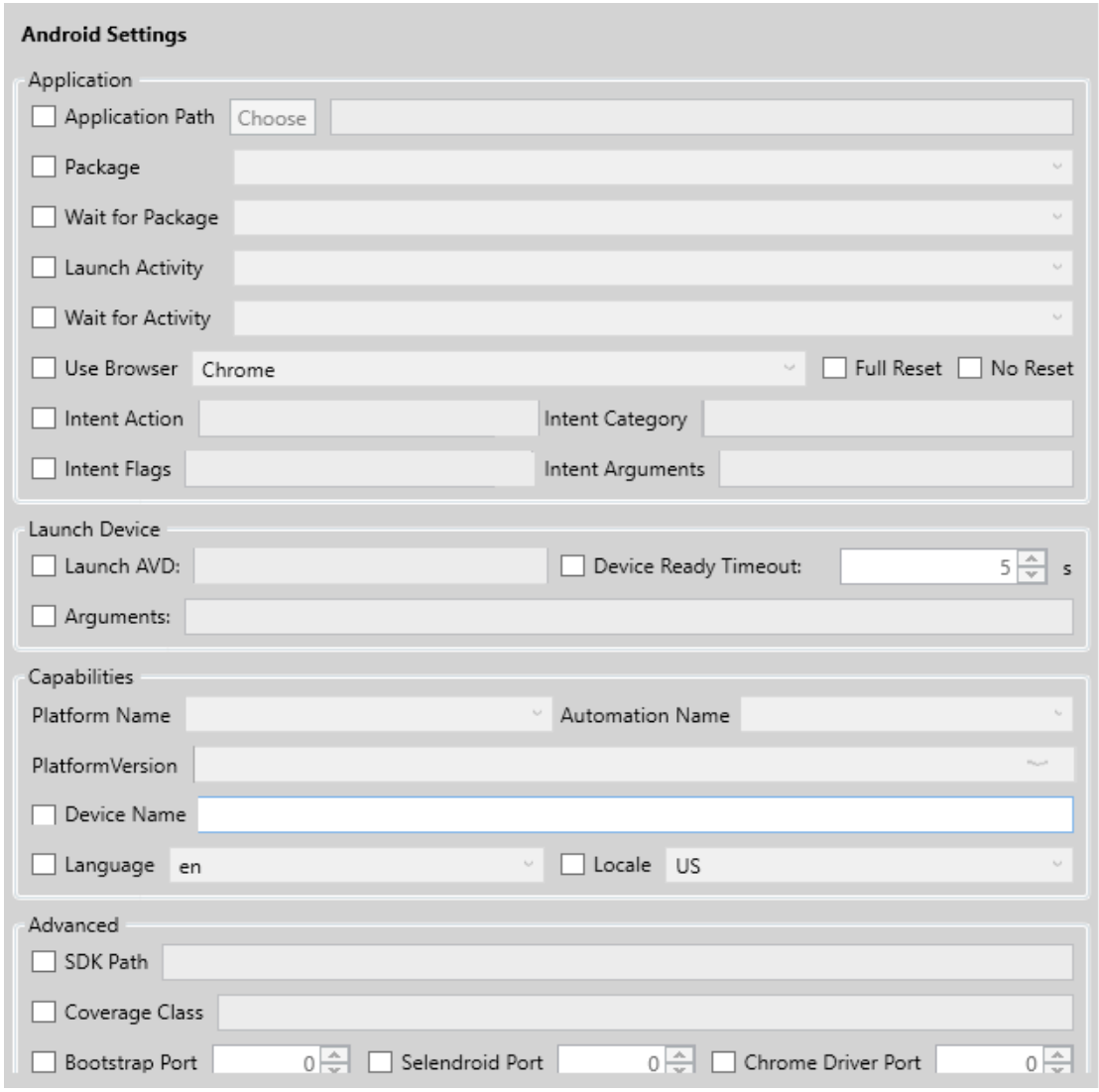

Figure 3.4.2.2 : Screen shot of Appium Android Settings

Figure 3.4.2.3 shows the General Settings page of Appium. On this page it is necessary to set server address and port.

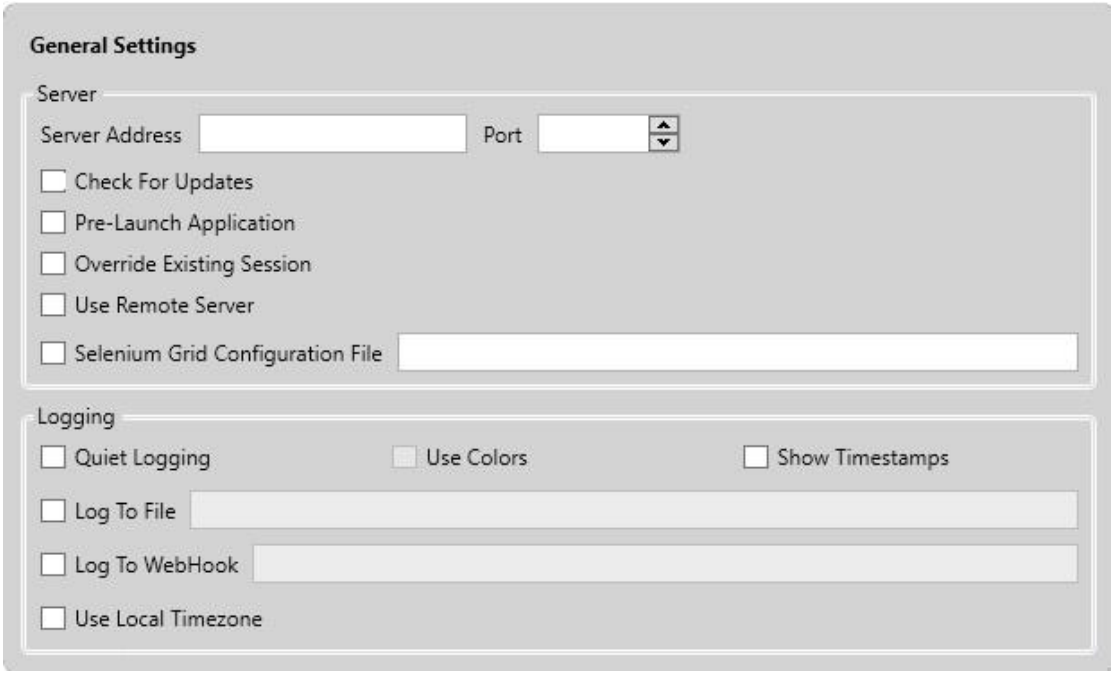

Figure 3.4.2.3 : Screenshot of Appium General Settings

C.27 Documentation

The number of Appium documents is not as much as Robotium documents. Especially, as setting up working environment is a long and complicated process, users need documentation that clearly describes this process; however, it is difficult to find such documentation. Mostly, finding necessary information may be possible using videos and tutorials prepared by users as a result of their experiences. Even Appium e-books and tutorials are not satisfactory. Thanks to the documents prepared by the experienced users, it is possible to indicate that, Appium documentation is helpful and adequate.

C.28 Community

As Appium is an open source framework and used by both Android and iOS testers, its community is big and active.

Appium has an active official discussion group [69] where Appium users may ask questions, open issues and discuss about Appium testing.

By April,2016, on its Github page[19] there are 740 open, 3.547 closed issues. On stackoverflow.com there are 1916 entry with ‘appium’ tag by April,2016 [56]. These numbers can be considered as indicators and we may say that Appium community is remarkably active.

3.4.3 UIAUTOMATOR

We have evaluated UIAutomator according to the criteria represented in Table 4.1. In this part, we explain the ability of UIAutomator to meet each of the criteria.

C.1 Insert Delays into Test Code

UIAutomator allows for simulating delays by inserting waits inside the code. Using `sleep(milliseconds)` method it is possible to simulate wait for specified time in milliseconds.

C.2 Testing Gestures

UIAutomator allows simulating most used user gestures partially.

- In order to simulate drag gesture following method can be used:

```
getUiDevice().drag(startX, startY, endX, endY, steps)
```

In this method the parameters ‘startX’ and ‘startY’ represents the coordinates of the point where drag action starts. The parameters ‘endX’ and ‘endY’ represents the coordinates of the point where drag action ends. The parameter ‘steps’ represents the number of steps to complete the drag action. More number of steps means a slower action.

- In order to simulate swipe gesture following method can be used:

```
getUiDevice().swipe(startX, startY, endX, endY, steps)
```

In this method the parameters ‘startX’ and ‘startY’ represents the coordinates of the point where swipe action starts. The parameters ‘endX’ and ‘endY’ represents the coordinates of the point where swipe action ends. The parameter ‘steps’ represents the number of steps to complete the swipe action.

- In order to simulate pinch in and out gesture following lines of code can be used:

```
UiObject.pinchIn(percent, steps);
UiObject. pinchOut(percent, steps);
```

pinchIn() and pinchOut() are the methods of UiObject class. Thus, in order to use these methods firstly an instance of UiObject class is created. Then, to simulate the gesture that two finger move towards other pinchIn() method, to simulate the gesture that two fingers move opposite across other pinchOut() method is used. The

parameter, ‘percent’ represents “percentage of the object's diagonal length for the pinch gesture”[70]. The parameter ‘steps’ represents the number of steps to complete the gesture.

- In order to simulate long click gesture, UIAutomator provides the method;

```
UiObject.longClick();
```

However, this method does not work properly. We have found that, the best way to simulate long click is using swipe() method [71].

```
getUiDevice().swipe(startX, startY, endX, endY, steps)
```

By giving the same x and y coordinates for starting and ending points in swipe method, it is possible to simulate long click.

C.3 Selecting Different Options At The Same Time (Undesired Multi Selection)

UIAutomator resources do not provide information if selecting different options at the same time is possible or not using the tool. During test development process, we could not find any method or function to simulate multiselection either.

C.4 Taking Screenshot

UIAutomator enables taking screenshot of the real or virtual device. Using following lines of code, it is possible to take screenshot:

```
File path = new File("file path");
geUiDevice().takeScreenshot(path);
```

In the first line of the code above, the path where the screenshot is stored is defined. Then, the screenshot is taken.

C.5 Reaching "Back", "Recent Apps" And "Home" Buttons

UIAutomator enables to reach and click Android hardware buttons.

- In order to click on Android Back button, `getUiDevice().pressBack()` method can be used.

- In order to click on Android Home button, `getUiDevice().pressHome()` method can be used.

- In order to click on Android Recent Apps button, `getUiDevice().pressRecentApps()` method can be used.

C.6 Interaction With Soft Keyboard

UIAutomator allows for reaching an clicking on soft keyboard buttons by using key codes of them as follows;

```
getUiDevice().pressKeyCode(key code);
```

However, the tool does not provide a method to hide the soft keyboard. Pressing back button may be a solution to hide the keyboard however sometimes when soft keyboard is not opened unexpectedly, clicking on back button results with an undesired action.

C.7 Different Internet Connection (No Connection, Wi-Fi, 3G, 2G) & C.8 Airplane Mode

UIAutomator enables for reaching nearly all UI elements. In order to edit connection and mode settings of the device, it is possible to go to 'Settings' page of the device and make necessary settings such as turning on/off Wi-Fi, turning on/off Airplane Mode, turning on/off mobile data. Unlike other automation tools, UIAutomator also allows changing the preferred network type to 2G or 3G

C.9 Interaction With Status Bar Notifications

UIAutomator partially allows interaction with status bar notification. Using the tool, it is possible to open notification area using the method;

```
getUiDevice().openNotification();
```

However, it is not possible to click on desired notification either by sending the coordinates x and y as parameters or creating a WebElement object for the notification.

C.10 Generating Test Report File

On discussion forums, it is stated that generating test reports is not possible using UIAutomator [72]. Command prompt shows execution results (execution time, success, failure) however, it does not provide a test report file as XML.

C.11 Record And Replay Property

On resources we searched [30] [73], it is stated that UIAutomator does not provide record and replay functionality.

C.12 Simulating An Incoming Call/Sms

UIAutomator resources do not provide information if simulating an incoming call is possible or not using the tool. When we examined the API, we could not find any method or function for simulating an incoming call or sms using UIAutomator.

C.13 Simulating Popup Alert (Alarm, Calendar)

UIAutomator resources do During test implementation, we could not find any method or function to simulate such an action either..

C.14 Simulating An Incoming Push Message From Another App

UIAutomator resources do not provide information if simulating an incoming push message is possible or not using the tool. When we examined the API, we could not find any method or function to simulate an incoming push message.

C.15 Simulating Inserting And Removing Charger

UIAutomator resources do not provide information if simulating inserting and removing charger is possible or not using the tool. When we examined the API, we could not find any method or function to simulate inserting and removing charger.

C.16 Testing The Behavior Of The App When The Battery Is Low

UIAutomator resources do not provide information if simulating low battery is possible or not using the tool. When we examined the API, we could not find any method or function to simulate low battery either.

C.17 Testing The Behavior Of The App When The Memory Is Low

UIAutomator resources do not provide information if simulating low battery is possible or not using the tool. When we examined the API, we could not find any method or function to simulate low memory either.

C.18 Testing On Emulator

UIAutomator enables running tests on emulator (virtual device). In our case study, we have run our test suite on emulator.

C.19 Testing On Real Device

It is clearly stated that running UIAutomator tests on real device is possible [74].

C.20 Testing With Apk File &&
C.21 Testing With Source Code

In order to test an application using UIAutomator, it is needed to select the target application under 'Applications' menu of the device. The only requirement for testing is, the application's being installed on the device. Thus, there is no difference between testing an application with APK file or source code. As a result, both testing with APK file and testing with source code is possible with UIAutomator.

C.22 Test Language

UIAutomator uses Java as the language for implementing the test code.

C.23 Executing Tests Using Command Prompt

Running UIAutoamator tests using command prompt is possible using the build tools Ant or Gradle. We have run our tests using Ant tool. In order to run a test suite using command prompt following steps should be followed [75]:

- Firstly, configuration file is created:

```
<android-sdk>/tools/android create uitest-project -n <name> -t
1 -p <path>
```

- Then, JAR file is created:

```
ant build
```

- The created JAR file is pushed to the device:

```
adb push ~/dev/workspace/Settings/bin/LaunchSettings.jar
/data/local/tmp/
```

- After pushing JAR file to the device, test is executed:

```
adb shell uiautomator runtest LaunchSettings.jar -c
com.my.LaunchSettings
```

C.24 Testing Hybrid Applications

On the resources we searched, it is stated that UIAutomator does not support hybrid applications [76][77]. In our case study, we have also observed that UIAutomatorviewer recognizes some webview objects but not all of them. For example, the webview buttons 'New Game' and 'Undo' could be reached without any problem. However, the main game elements that represent numbers could not be recognized by UIAutomatorViewer and could not be reached by UIAutomator. We have concluded that, testing hybrid applications is partially enabled by UIAutomator.

C.25 Need To Have Developer Level Knowledge About The Code

UIAutomator enables to develop and run test suite with minimal knowledge about implementation of the application. Using UIAutomatorviewer, the properties such as id, index, class name, package name of each UI component can be accessed easily. Using these properties, test cases can be written and run without additional information.

C.26 Working Environment Setup

As the tool comes with Android Test Library, it is very easy and fast to setup UIAutomator working environment. In order to create tests using UIAutomator, android.jar and uiautomator.jar files and Junit4 library should be imported to the test project.

C.27 Documentation

One of the main disadvantages of UIAutomator is lack of documentation. During our study, we have faced difficulties for finding documents about the tool. Especially in compared to the other tools we have studied, it is obvious that there is not enough available documentation about UIAutomator.

C.28 Community

By March 2016 on its Github page [78] there are 54 open 94 closed issue. On stackoverflow.com [56] there are 349 entries with the tag ‘uiautomator’. Compared to Robotium and Appium these numbers are considerably low.

We have also experienced that, when we opened an issue on GitHub or stackoverflow.com, either we did not receive feedback or it takes a long time (weeks) to receive.

Thus, we have concluded that, the community of the tool cannot be considered as active.

CHAPTER IV

4 EXPERIMENTAL STUDY

In this chapter we explained the case study that we have conducted using two Android applications and the selected mobile application testing tools. In our case study we have developed test code for applications using test automation tools. With each of the tool we have created the same test suite. This allows us to compare the performance of the tools and efforts needed to be spent to create and run test cases for each of the tool.

4.1 PREPARATION

For our case study we used two Android applications; Droidweight and 2048. These are open source applications whose source code is available on GitHub. Details of these applications are explained in section 4.2.

We have used Eclipse Mars 4.5.0 and JDK 1.8.0_72 as development environment. We have performed our study on Intel Core i5 processor of a 2. 20 GHz clock with 64 bit Windows 10 operating system. We have downloaded and installed Android SDK [79] which includes development tools, emulators and necessary libraries to build Android applications. We have run all applications and tests using virtual device, whose properties are as follows:

- Device: Nexus 5 (4.95'', 1080 x 1920: xxhdpi)
- Target: Google APIs (Google Inc.) – API Level 23
- CPU/ABI: Google APIs Intel Atom (x86)
- Keyboard: Hardware keyboard present
- Skin: No skin
- Front Camera: None
- Back Camera: None
- RAM: 2048
- VM Heap:64
- Internal Storage : 200 MiB
- SD Card Size: 200 MiB
- Emulation Options: Use host GPU

We have used Robotium version 5.5.4. For this purpose, we have downloaded [80] the jar file; 'robotium-solo-5.5.4.jar' and imported to our test projects in Eclipse.

We have downloaded [81] and used Appium version 1.4.16.1. After downloading Appium server, we have made necessary settings as follows:

On Android Settings window:

- Platform Name: Android
- Automation Name: Appium
- Platform Version: 6.0 Marshmallow (API Level 23)
- Device Name: MyEmulator

On General Settings window:

- Server Address: 127.0.0.1
- Port: 4723

In order to use UIAutomator, we have imported the jar files; 'android.jar' and 'uiautomator.jar' from our Android SDK. We have also added 'Junit4' library to our test projects.

4.2 TEST CASES

We have used equivalence class partitioning to generate tests for our applications. Equivalence class partitioning is a testing technique whose idea is to divide a set of tests into partitions which are expected to be responded by the system under test equivalently. [82]. In this technique at least one condition from each partition is tested and it is expected that all conditions in the same partition behave in the same way.

Table 4.2.1 shows the equivalent classes that we have used to create test suite in our case study. We have defined these equivalent classes based on our criteria set. By these equivalent classes, we have defined possible behaviors of the application under 8 parts. We have defined 22 equivalent classes.

Table 4.2.1: EQUIVALANCE CLASSES

		Equivalance Class
Connection	**A1**	Airplane mode on
	A2	Airplane mode off, Wi-Fi off, Mobile data off
	A3	Airplane mode off, Wi-Fi on, Mobile data off
	A4	Airplane mode off, Wi-Fi off, mobile data on
	A5	Airplane mode off, Wi-Fi on, Mobile data

		on
Editview	**E1**	Editview is clickable
	E2	Editview is editable
	E3	Editview is disabled
Button	**B1**	Button is clickable
	B2	Button is disabled
Soft keyboard	**S1**	Softkeyboard is clickable
	S2	Softkeyboard is hiden
	S3	Softkeyboard is disabled
Notification area	**N1**	Notification area is pulled down
	N2	Notification area is clickable
	N3	Notification area is unreachable
Android BACK button	**C1**	Android BACK button is clickable
	C2	Android BACK button is disabled
Android HOME button	**H1**	Android HOME button is clickable
	H2	Android HOME button is disabled
Android RECENT APPS button	**R1**	Android RECENT APPS button is clickable
	R2	Android RECENT APPS button is disabled

In addition to these test cases, for evaluating gesture abilities, we choose a set of points for drag, swipe, long click, pinch, and click on two widgets actions. This results in five test cases to check gesture abilities. Inserting delays and taking screenshots are also tested in two test cases.

Our test cases for the application DroidWeight are as follows. As the other test cases use the same framework utilities we have not implemented all of the functionalities and combinations.

- A4: We have set airplane mode off, Wi-Fi off and Mobile data on.
- E1: We have clicked on editviews whose ids are; ‘currentWeight’, ‘dialoginput’ and ‘comment’.
- E2: We have edited the text in editviews whose ids are: ‘currentWeight’, ‘dialoginput’ and ‘comment’.
- B1: We have clicked on buttons whose ids are; ‘button1’ and ’calculator’
- S1: We have clicked on soft keyboard buttons; ‘t’ ,‘e’,’s’,’t’ and ‘delete’.
- S2: We have hidden the soft keyboard.
- N1: We have opened the notification area.
- N2: We have clicked on a notification whose coordinates are (521,349).

- C1: We have clicked on Android Back button on ‘Statistics’ page.
- H1: We have clicked on Android Home button on ‘Statistics’ page.
- R1: We have clicked on Android Recent Apps button on ‘Statistics’ page.

In order to find the ability of the tool for disabling UI elements and test their functionality in disabled mode, we have also created test cases.

- E3: We have disabled the editview with id ‘currentWeight’ and clicked on it.
- B2: We have disabled the button with text ‘Go’ and clicked on it.
- S3: We have disabled the softkeyboard when entering input to the edit view with id ‘comment’
- N2: We have disabled the notification area to be opened
- C3: We have disabled the Android ‘Back’ button and clicked on it.
- H2: We have disabled the Android ‘Home’ button and clicked on it.
- N3: We have disabled the Android ‘Recent Apps’ button and clicked on it.

Additionally,

- We have selected the following representors from gestures;
    - We have dragged from the point (458,1424) to the point (458,590) on ‘History’ page.
    - We have swiped from the point (458,1424) to the point (458,590) on ‘History’ page.
    - We have click long on an item whose id is ‘LinearLayout01’
    - We have pinch with two fingers on ‘History’ page. The first finger moves from the point (541,781) to (541,590). The second finger moves from the point (541,1072) to (541,1276).
- We have taken screenshot of the ‘History’ page.
- We have clicked on texts ‘History’ and ‘Statistics’ simultaneously.
- We have inserted waits after clicking on buttons, performing drag gesture and opening notification area.

As DroidWeight and 2048 do not have any image, it is not meaningful but possible to write test code for pinch gesture. So, we have implemented test code for pinch gesture even there is no action mapped to it.

Our test cases for the application 2048 are:

- A4: We have set airplane mode off, Wi-Fi off and Mobile data on
- B1: We have clicked on buttons whose texts are ‘New Game’ and ‘Cancel’
- N1: We have opened the notification area.
- N2: We have clicked on a notification whose coordinates are (521,349)
- C1: We have clicked on Android Back button on ‘main page

- H1: We have clicked on Android Home button on ‘main page
- R1: We have clicked on Android Recent Apps button on ‘main page

The application 2048 does not include an object with editview type. As this widget does not exist, we could not be able to write a test case, which is expected, for editview. Softkeyboard is a widget which can be opened and used when an editview is clicked. As an editview does not exist on 2048, we could not be able to write a test case that tests the behavior of the softkeyboard either.

Additionally,

- We have dragged an item whose text is ‘2’ and index is ‘0’ to the right.
- We have taken screenshot of the main page
- We have clicked on the buttons ‘New Game’ and ‘Undo’ simultaneously.
- We have inserted waits after clicking on buttons and performing drag gesture and opening notification are.
- We have pinch with two fingers on main page. The first finger moves from the point (541,781) to (541,590). The second finger moves from the point (541,1072) to (541,1276).

The application does not include an item that has any functionality triggered by long click or swipe. So, it is not functional and meaningful to write test code that simulates these gestures. As we have implemented a functional test code for these gestures for the application ‘DroidWeight’, we have not implemented a test case to simulate long click and swipe for the application 2048.

4.3 EXPERIMENT

We present our evaluation of the tools under two sections: test code implementation aspect (criteria C1-C9) and the criteria where we do not have to implement test code (criteria C10-C28).

For the evaluation of the criteria between C1-C9, we have developed test suites that include test cases whose details are explained in section 4.2. Using each of the tool, Robotium, Appium and UIAutomator, we have implemented test suites for the applications ‘DroidWeight’ and ‘2048’ whose details are explained in the section 3.2.

4.3.1. Robotium results

For the application ‘DroidWeight’ we have developed a test suite consisting of 9 test cases, using Robotium. As Robotium gives an alphabetic order to run tests, we have given test case names starting with a letter from ‘A’ to ‘I’ to set execution order. We reached UI elements either by their id or text.

Table 4.3.1.1 shows the test cases implemented using Robotium, the equivalent classes they include and functionalities they test.

Table 4.3.1.1: ROBOTIUM TEST CASES

Test Case	**Equivalent Class or Functionality**
testAInternetConnection	A4
testBScreenItems	E1,E2,B1
testCLongClick	Long click gesture
testDDrag	Drag gesture
testESwipe	Swipe gesture
testFScreenShot	Taking screenshot
testGSoftKeyboard	S1,S2
testHWButtons	C1,H1,R1
testIPinch	Pinch gesture

Total line of code for implementing these tests with Robotium is 119. Total time to execute test cases is 188,422 seconds. Figure 4.3.1.1 shows the test report that is generated after the execution of the test suite. In this report it is shown that there are 9 tests in the class (lines 4-12). 9 of them is started and 0 of them is ignored. There is no failure in the executed test suite. As reaching notification area and simulating simultaneous multi selection is not possible using Robotium, we could not create test cases for this functionalities,

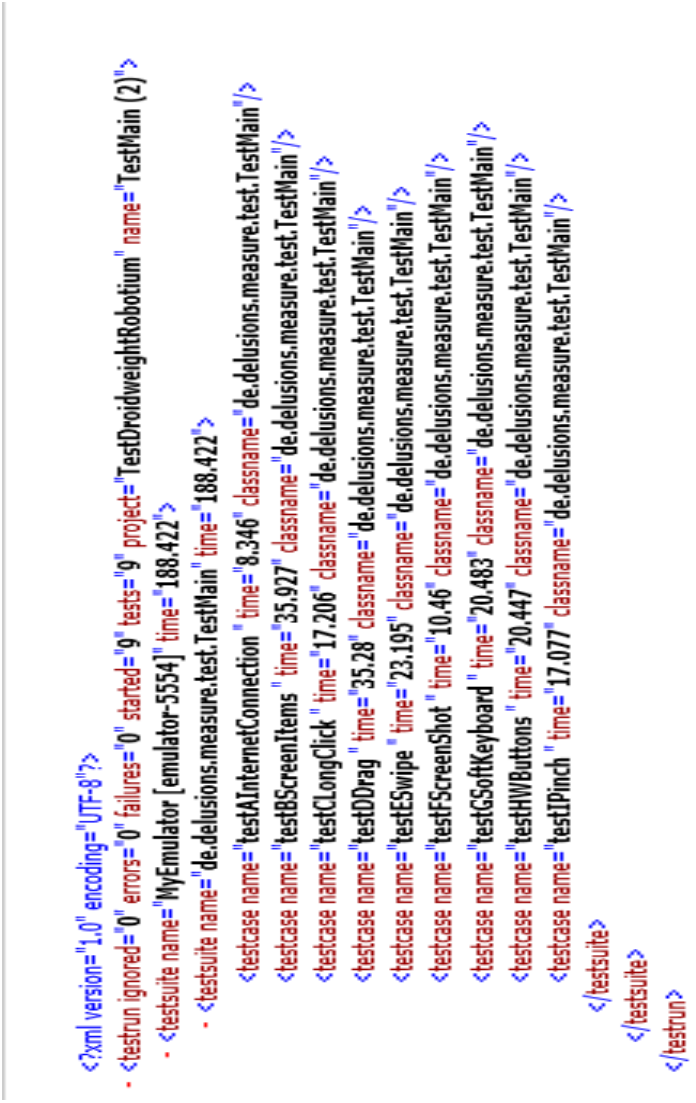

Figure 4.3.1.1: Robotium Test Report of DroidWeight

The report in Figure 4.3.1.1 shows each test case by its name and execution time. The report generated by Robotium shows

- Execution of the test case ‘testAInternetConnection’ in which we set airplane mode and connection settings lasts 8,346 seconds with success.
- Execution of the test case ‘testBScreenItems’ in which we test UI elements by clicking on buttons, clicking on and editing editviews, and inserting waits, lasts 35,927 seconds with success.
- Execution of the test case ‘testCLongClick’ in which we simulate long click gesture, lasts 17,2016 seconds with success.
- Execution of the test case ‘testDDrag’ in which we simulate drag gesture, lasts 35,28 seconds with success.

- Execution of the test case ‘testESwipe’ in which we simulate swipe gesture lasts 23,195 seconds with success.
- Execution of the test case ‘testFScreenShot’ in which we take screenshot of the device, lasts 10,46 seconds with success.
- Execution of the test case ‘testGSoftKeyboard’ in which we test entering input using soft keyboard lasts 20,483 seconds with success.
- Execution of the test case ‘testHWButtons’ in which we test Android hardware buttons; Back, Home and Recent Apps by clicking on each, lasts 20,447 seconds with success.
- Execution of the test ‘testIPinch’ in which we simulate pinch gesture with two fingers, lasts 17,077 seconds with success.

Additionally, we have created test cases that disable the UI items and reach them. Robotium does not provide functionality to disable the notification area and Android hardware buttons. So, we could not be able to write cases that test the equivalent classes; N2, C2, H2 and N3. We have generated 3 test cases that test the UI elements in disabled mode. Figure 4.3.1.2 shows the report generated after execution of these 3 test cases. The report shows that, we get error when we try to reach a disabled element.

The report represents that,

- Execution of the test case ‘testADisabledEditView’ which includes the equiavalent class E3, lasts 7,477 seconds and ends with an error.
- Execution of the test case ‘testBDisabledButton’ which includes the equivalent class B2, lasts 10,241 seconds and ends with an error.
- Execution of the test case ‘testCDisabledKeyboard’ which includes the equivalent class S3, lasts 12,023 seconds and ends with an error.

Figure 4.3.1.2: Robotium Report Of DroidWeight Test Cases With Disabled UI Elements

For the application 2048, we have created a test suite consists of 6 test cases using Robotium. Inside these test cases we have tested the application behaviors stated in the section 4.2. Total line of code is 68. Total time to execute test suite is 93,397 seconds.

After running the test suite, the tool has generated a test report which is shown in Figure 4.3.1.3.

```
<?xml version="1.0" encoding="UTF-8"?>
- <testrun ignored="0" errors="0" failures="0" started="6" tests="6" project="2048-androidTestRobotium" name="TestMain (1)">
  - <testsuite name="MyEmulator [emulator-5554]" time="120.441">
    - <testsuite name="com.uberspot.a2048.test.TestMain" time="120.441">
        <testcase name="testAInternetConnection " time="15.55" classname="com.uberspot.a2048.test.TestMain"/>
        <testcase name="testBScreenItems " time="25.112" classname="com.uberspot.a2048.test.TestMain"/>
        <testcase name="testDDrag " time="23.074" classname="com.uberspot.a2048.test.TestMain"/>
        <testcase name="testFScreenShot " time="25.363" classname="com.uberspot.a2048.test.TestMain"/>
        <testcase name="testHWButtons " time="14.333" classname="com.uberspot.a2048.test.TestMain"/>
        <testcase name="testIPinch " time="17.009" classname="com.uberspot.a2048.test.TestMain"/>
      </testsuite>
    </testsuite>
</testrun>
```

Figure 4.3.1.3: Robotium Test Report of 2048

The test report generated by Robotium shows that there are 6 tests in the test suite. As reaching the notification area and simultaneous multi selection is not supported by Robotium, the test suite consists of 6 test cases instead of 8 test cases.6 of them is started and 0 of them is ignored. There is no failure in the executed test suite.

The report shows each test cases by their name and execution time. The report in Figure 4.3.1.3 shows that;

- Execution of the test case ‘testAInternetConnection’ in which we set airplane mode and connection settings lasts 13,051 seconds with success.
- Execution of the test case ‘testBScreenItems’ in which we test UI elements by clicking on buttons, and inserting waits, lasts 23,343 seconds with success.
- Execution of the test case ‘testDDrag’ in which we simulate drag gesture, lasts 19,415 seconds with success.
- Execution of the test case ‘testFScreenShot’ in which we take screenshot of the device, lasts 22,644 seconds with success.
- Execution of the test case ‘testHWButtons’ in which we test Android hardware buttons; Back, Home and Recent Apps by clicking on each, lasts 14,945 seconds with success.
- Execution of the test ‘testIPinch’ in which we simulate pinch gesture with two fingers, lasts 17,077 seconds with success.

While implementing the test cases using Robotium, we have reached UI elements by their id or their text. In order to reach id of the elements we used ‘uiautomatorviewer’. As 2048 is a hybrid project that includes a webview, we could not be able to reach properties such as id and index of the UI elements of that application. Thus, we have reached all UI elements of 2048 by their text.

4.3.2. Appium results

We have developed test code for DroidWeight and 2048 using Appium. In order to set execution order of test cases, we give priority to each test case and define test case names starting with numbers from 1 to 11.

For DroidWeight, we have developed a test suite consists of 11 test cases. As a result we have implemented a test suite of 166 LOC for DroidWeight application. If we exclude the lines that are written to test notifications and multi selection, we see that 150 LOC is written for 9 test cases which are the same test cases with Robotium. So, we have concluded that more LOC is needed to implement the same test suite using Appium than using Robotium.

Table 4.3.2.1 shows the test cases implemented using Appium, the equivalent classes they include and functionalities they test.

Table 4.3.2.1: APPIUM TEST CASES

Test Case	**Equivalent Class or Functionality**
test11ZInternetConnection	A4
test1BScreenItems	E1, E2, B1
test2CLongClick	Long click gesture
test3DDrag	Drag gesture
test4ESwipe	Swipe gesture
test5FScreenShot	Taking screenshot
test6GSoftKeyboard	S1, S2
test7MultiTouch	Simultaneous multi touch
test8HWButtons	C1, H1, R1
test9INotifications	N1,N2
test10KPinch	Pinch gesture

After running the test suite, Appium generates a report which is presented in Figure 4.3.2.1. The report shows that, total execution time is 273163 milliseconds. 11 test cases are passed and 0 test cases are failed. Each test case is shown with its name and execution time. The report generated by Appium in Figure 4.3.2.1 shows that

- Execution of the test case ‘test10ZInternetConnection’ in which we set airplane mode and connection settings lasts 38718 milliseconds with success.
- Execution of the test case ‘test1BScreenItems’ in which we test UI elements by clicking on buttons, clicking on and editing editviews, and inserting waits, lasts 41996 milliseconds with success.
- Execution of the test case ‘test2CLongClick’ in which we simulate long click gesture, lasts 6081 milliseconds with success.

- Execution of the test case ‘test3DDrag’ in which we simulate drag gesture, lasts 12861 milliseconds with success.
- Execution of the test case ‘test4ESwipe’ in which we simulate swipe gesture lasts 59614 milliseconds with success.
- Execution of the test case ‘test5FScreenShot’ in which we take screenshot of the device, lasts 9560 milliseconds with success.
- Execution of the test case ‘test6GSoftKeyboard’ in which we test entering input using soft keyboard lasts 30178 milliseconds with success.
- Execution of the test case ‘test7MultiTouch’ in which we simulate two simultaneous touch on the screen, lasts 3002 milliseconds with success.
- Execution of the test case ‘test8HWButtons’ in which we test Android hardware buttons; Back, Home and Recent Apps by clicking on each, lasts 13564 milliseconds with success.
- Execution of the test case ‘test9INotifications’ in which we open the notification area and click on a notification; lasts 5375 milliseconds with success.
- Execution of the test ‘testIPinch’ in which we simulate pinch gesture with two fingers, lasts 24224 milliseconds with success.

Appium does not provide any functionality to disable UI elements. That is why, we could not be able to implement test cases that include equivalent classes; E3,B2,S3,R2,C2,H2 and N2.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Default suite						
Default test	11	0	0	273.163		

Class	Method	Start	Time (ms)
Default suite			
Default test — passed			
de.delusions.measure.test.TestMainAppnumDroidweight	test10INotifications	1461534980164	5375
	test11ZInternetConnection	1461534985541	38718
	test1BScreenItems	1461534779019	41996
	test2CLongClick	1461534821046	6081
	test3DDrag	1461534827130	12861
	test4ESwipe	1461534840000	59614
	test5FScreenShot	1461534899618	9560
	test6GSoftKeyboard	1461534909184	30178
	test7MultiTouch	1461534939365	3002
	test8IPinch	1461534942370	24224
	test9HWButtons	1461534966597	13564

Figure 4.3.2.1: Appium Test Report of DroidWeight

For the application 2048, we have implemented a test suite that consists of 8 test cases that test the application behavior stated in section 4.2. Total execution time for these test cases is 156077 milliseconds.

We have written 113 LOC for this test suite. When we exclude the lines written for the notification area and multi touch, there are 97 LOC. This number is more that the LOC of the same test cases written by Robotium.

After running the test suite, the tool provides a report which is presented in Figure 4.3.2.2. The report shows that, 8 test cases are passed and 0 test cases are failed. According to the report in Figure 4.3.2.2 generated by Appium;

- Execution of the test case 'test7AInternetConnection' in which we set airplane mode and connection settings lasts 45292 milliseconds with success.
- Execution of the test case 'test1BScreenItems' in which we test UI elements by clicking on buttons, and inserting waits, lasts 31209 milliseconds with success.
- Execution of the test case 'test2DDrag' in which we simulate drag gesture, lasts 10858 seconds with failure.
- Execution of the test case 'test3FScreenShot' in which we take screenshot of the device, lasts 15758 milliseconds with success.
- Execution of the test case 'test5HWButtons' in which we test Android hardware buttons; Back, Home and Recent Apps by clicking on each, lasts 2648 milliseconds with success.
- Execution of the test case 'Test6IMultiTouch' in which we test clicking on two buttons at the same time, lasts 16330 milliseconds with success.
- Execution of the test case 'test6INotification' in which we open the notification area and click on a notification lasts 5327 milliseconds with success.
- Execution of the test 'testIPinch' in which we simulate pinch gesture with two fingers, lasts 15162 milliseconds with success.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Default suite						
Default test	8	0	0	156.077		

Class	Method	Start	Time (ms)
Default suite			
Default test — passed			
com.uberspot.a2048.test.TestMainAppium	test1BScreenItems	1461560193010	31209
	test2DDrag	1461560224225	10854
	test3FScreenShot	1461560235082	15758
	test4Pinch	1461560250843	15162
	test5HWButtons	1461560266008	2648
	test7AInternetConnection	1461560276383	45292
	test7Notifications	1461560321678	5327
	test6IMultiTouch	1461560322375	16330

Figure 4.3.2.2: Appium Test Report of 2048

Similar to Robotium, we have reached UI components by their id and text using Appium. For DroidWeight, we could easily reach the elements. Even, we have faced difficulties, we have reached WebView elements of 2048 using Appium.

4.3.3. UIAutomator results

After Robotium and Appium, we have developed test code for DroidWeight and 2048 using UIAutomator. Developing and running tests using UIAutomator needs more time and effort. Unlike Appium and Robotium, we do not select a target project to test when we create a UIAutomator test project. When we create test projects with Appium or Robotium using Eclipse, at the beginning of the test project creation, we need to select the target application that we aim to test. This means, only one application can be reached and tested inside a test project with these tools. On the other hand, UIAutomator test projects does not require a target application. Inside the test project, we may navigate any application or setting using UIAutomator. This situation makes the entire device menu reachable. On the other hand, it increases the line of code needed for developing tests. Moreover, running UIAutomator tests requires some time to spend with command prompt. As it is explained in section 3.4.3, a number of commands are used for each time to execute a test suite. These commands are for creating test project, building it, pushing the test project to device and running the tests. In addition to these commands it is also necessary to add another command to clear the application data. Because, when a test suite runs, UIAutomator stores data in cache and when we modify the test suite and try to run the modified suite, the tool may run the previous test suite from the cache. Thus, we

have concluded that executing test suite is a time consuming and painful process with UIAutomator.

For DroidWeight, we have developed a test suite consists of 10 test cases instead of 11, because simulating simultaneous multi selection is not possible with UIAutomator. Even it is possible to open the notification area, we have experienced that, UIAutomator does not allow for clicking on a notification.

We have written 131 LOC for DroidWeight application. If we exclude the lines which is written to test notifications, we see that 127 LOC is written for 9 test case which are the same test cases with Robotium. So, we have concluded that for the same test suite, UIAutomator requires more LOC than Robotium and less LOC than Appium.

Table 4.3.3.1 shows the test cases implemented using UIAutomator, the equivalent classes they include and functionalities they test.

Table 4.3.3.1: UIAUTOMATOR TEST CASES

Test Case	Equivalent Class or Functionality
testAInternetConnection	A4
testBScreenItems	B1,B2
testCLongClick	Long click gesture
testDDrag	Drag gesture
testESwipe	Swipe gesture
testFScreenShot	Taking screenshot
testGSoftKeyboard	S1
testHWButtons	C1, H1, R1
testINotification	N1
testKPinch	Pinch gesture

UIAutomator prints the test results on command prompt. However, it does not provide a report file. So, we have examined the execution times on command prompt. We have found that, total execution time for 9 test cases is 160,52 seconds. We have also run each test cases individually. We have found that;

- Execution of the test case ‘testAInternetConnection’ in which we set airplane mode and connection settings lasts 41,460 seconds.
- Execution of the test case ‘testBScreenItems’ in which we test UI elements by clicking on buttons, and inserting waits, lasts 43,798 seconds.
- Execution of the test case ‘testCLongClick’ in which we simulate long click gesture, lasts 21,477 seconds.

- Execution of the test case ‘testDDrag’ in which we simulate drag gesture, lasts 22,651 seconds.
- Execution of the test case ‘testESwipe’ in which we simulate swipe gesture lasts 19,760 seconds.
- Execution of the test case ‘testFScreenShot’ in which we take screenshot of the device, lasts 1,514 seconds.
- Execution of the test case ‘testGSoftKeyboard’ in which we test entering input using soft keyboard lasts 15,276 seconds.
- Execution of the test case ‘testHWButtons’ in which we test Android hardware buttons; Back, Home and Recent Apps by clicking on each, lasts 13,741 seconds
- Execution of the test case ‘testINotification’ in which we open the notification area lasts 1,203 seconds.
- Execution of the test ‘testIPinch’ in which we simulate pinch gesture with two fingers, lasts 9,85 seconds with success.

UIAutomator does not provide any functionality to disable UI elements. . That is why, we could not be able to implement test cases that include equivalent classes; E3, B2, S3, R2, C2, H2 and N2.

UIAutomatorviewer provides a number of information of UI elements such as class name, resource id, bounds, is clickable, is checkable. Reaching UI elements easily using one or more of these properties is possible. We have reached UI elements using class name, text, id and description of them.

Even manuals state that UIAutomator does not support hybrid applications, we have created a test project for 2048, in order to validate this statement. We have seen that the test cases, which are not related to the webview works well using UIAutomator. These test cases are;

- ‘testAInternetConnection’ in which we set airplane mode and connection settings lasts 22,410 seconds.
- ‘testFScreenShot’ in which we take screenshot of the device, lasts 1,256 seconds.
- testHWButtons’ in which we test Android hardware buttons; Back, Home and Recent Apps by clicking on each, lasts 4,743 seconds.
- ‘testINotification’ in which we open the notification area lasts 0,782 seconds.
- ‘testIPinch’ in which we simulate pinch gesture with two fingers, lasts 4,239 seconds

Moreover, some buttons inside the webview is also reachable. These are the elements that are recognized by UIAutomatorViewer. We could be able to reach and click on them with the following test case which is the same with the ‘testScreenItems’ test cases of Robotium and Appium test suites.

- ‘test1BScreenItems’ in which we test UI elements by clicking on buttons, and inserting waits, lasts 9,384 milliseconds.

On the other hand, there are some elements, which are the main draggable game elements that represent numbers such as ‘2’, ‘4’, ‘8’ and cannot be recognized by UIAutomatorViewer. We have reached these elements using their texts and performed drag gesture on them in Robotium and Appium test cases. However, UIAutomator does not allow reaching them. As a result of this experimental study, we have concluded that, UIAutomator partially supports testing hybrid applications.

4.3.4 Comparison on criteria C10-C28

For the criteria C10, test report generation, we have gathered information from manuals, blogs and forums and validated them with our experimental study. We have created test reports presented above, using Robotium and Appium. These are simple HTML reports which are informative and easy to understand. Appium allows generating reports with different types. The report type we used ‘emaillable-report’ is easier to read with its colored format in compared to Robotium. However, the values of ‘Start’ column is not easy to understand. It presents the starting time of the execution as Epoch time and it is needed to use a converter to convert this time to an understandable format. Epoch time is also known as Unix Time in which the time is “defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970” [83].

The criteria C11, which is about record and replay ability of the tools, we have gathered from the manuals the information that Robotium and UIAutomator does not support this property whereas Appium does. When we try to record and replay tests using Appium Inspector and make a deeper search, we have found that, this property is only supported on iOS platform. As we work on Windows, we could not be able to experience the record and replay property of Appium.

For the evaluation of the criteria between C12-C17 which is related to interrupt testing and leakage testing, we have searched on manuals and blogs. As this criteria is not supported by Robotium, Appium and UIAutomator, we have not included test cases for them.

The criteria between C18-C24 we have gathered information from manuals and supported a number of them with our experimental study. We have run our tests on emulator and using source code of the applications. On the other hand, the resources have provided sufficient information about running tests on real device and using apk file of the application.

Robotium and UIAutomator allows writing test cases using Java and Appium allows using any WebDriver compatible language. We have developed all of test suites

using Java. The notations are simple and self-explanatory. On the other hand, allowing different languages is an important advantage for Appium.

Each of the three tools enables executing tests using command prompt. For Robotium, running tests using command prompt is as easy as running tests using IDE. However, Appium and UIAutomator require build tools such as Ant or Gradle to run tests using command line. It needs an important amount of time and effort to execute tests using command line for these two tools. For Appium, command line is not a fast and easy way to execute tests, thus we do not prefer this way. However, UIAutomator allows execution only using command line. Thus, we have run all UIAutomator test suite with this way which is not an efficient one.

It is stated in the manuals that testing hybrid applications is possible with Robotium and Appium but not possible using UIAutomator. We have created test suite for the hybrid application 2048 using each tool. We have not experienced difficulties by Robotium. Appium supports hybrid applications as well but sometimes the tool may fail at recognizing webview elements. For example the main game elements that represents numbers such as '2', '4', '8', can be recognized and reached when a test suite is executed. Next time the same test suite is executed; these elements may not be reached by Appium. We have also created a test suite for the hybrid application 2048 using UIAutomator. We have experienced that, some webview elements can be reached by UIAutomator, where some of them cannot. So, we have concluded that, UIAutomator partially supports hybrid applications. As a result of this exploratory study, we have decided that, Robotium is the one which may test hybrid applications as well as native Android applications.

We have evaluated the criteria between C25- C28 as a result of our experiences. As Robotium, Appium and UIAutomator are black box testing tools, it was stated in manuals that knowledge about source code of the application is not necessary to create tests using these tools. In our experimental study, we have concluded that even this statement is generally true, some information such as package name and the launcher activity name about the source code is necessary for testing with Robotium. In addition, reaching and modifying the Android.Manifest file which provides information about an application to Android system [84], is required to use some Robotium methods such as taking screenshot and changing network state. For this methods, it necessary to give permission in Android.Manifest file of the application.

We have experienced that, setting up working environment such as downloading and installing necessary software, importing libraries and jar files, was fast and easy for the tools, Robotium and UIAutomator. However, Appium requires a number of steps whose details are explained in section 3.4.2 before starting test suite creation.

As Android application testing is a new area we had some doubts about documentation and community of the automation tools. However, even the number of official documentation such as books, manuals is few in numbers; blogs and

forums are highly active and provide most of the information about Robotium and Appium. When we need information that we could not find or when we face problems about the tool, we have created issues on active community Web pages and generally, got feedback in 1-7 days. However, this case is different for UIAutomator. The documents and tutorials that enable users to learn UIAutomator and its capabilities are insufficient. Moreover, getting feedback to an opened issue is either not possible or takes weeks. We concluded that, one of the main disadvantages of UIAutomator is lack of documentation and an active community.

CHAPTER V

5 RESULTS AND CONCLUSION

In this chapter, we present the results and conclusion of our study. In section 5.1 we present the results gathered from our comparative study. In section 5.2 we explain the limitations of this study. In section 5.3 we present our concluding remarks and suggestions for future work.

5.1. RESULTS

We represent the results of our comparative study on Robotium, Appium and UIAutomator in Table 5.1.1. In this table,

- "Yes" indicates that, it is explicitly stated that the tool meets the stated criteria.
- "No" indicates that, it is explicitly stated that the tool does not meet the stated criteria.
- "NI" indicates that, there is not any information if the tool meets the stated criteria or not.
- "Partially" indicates that, the tool does not completely but partially meets the stated criteria.

Table 5.1.1: RESULTING TABLE OF COMPARISON OF THE TOOLS WITH RESPECT TO THE CRITERIA

	Criteria	**Robotium**	**Appium**	**UIAutomator**
C1	Can we test delays	Yes	Yes	Yes
C2	Can we test gestures	Yes	Yes	Yes
C3	Can we select different options at the same time (undesired multi selection)	NI	Yes	NI
C4	Can the tool take screenshot	Yes	Yes	Yes
C5	Is it possible to reach "back", "recent apps"	Partially	Yes	Yes

	and "home" buttons using the tool			
C6	Is interaction with soft keyboard available	Yes	Yes	Partially
C7	Can we test the behavior of the app with different internet connection (no connection, Wi-Fi, 3G, 2G)	Partially	Partially	Yes
C8	Can we test behavior of the app on airplane mode	NI	Yes	Yes
C9	Is interaction with status bar notifications available	No	Yes	Partially
C10	Is it possible to generate test report files using the tool	Yes	Yes	No
C11	Does the tool have record and replay property	No	Partially	No
C12	Can we test the behavior of the app when a there is an incoming call/sms	No	No	NI
C13	Can we test the behavior of the app when a there is a popup alert (alarm, calendar)	NI	NI	NI
C14	Can we test the behavior of the app when a there is an incoming push message from another app	NI	NI	NI
C15	Can we test if inserting and removing charger causes any problem or not	NI	NI	NI
C16	Can we test the behavior of the app when the battery is low	NI	NI	NI
C17	Can we test the behavior of the app when the memory is low	NI	NI	NI
C18	Can we test on emulator	Yes	Yes	Yes
C19	Can we test on real device	Yes	Yes	Yes
C20	Is testing with apk file available	Yes	Yes	Yes
C21	Is testing with source code available	Yes	Yes	Yes
C22	Test language	Java	Any	Java
C23	Can we execute tests using command prompt	Yes	Yes	Yes
C24	Can we test hybrid applications	Yes	Yes	Partially
C25	Is it possible to test an application without developer level knowledge about the application code	Partially	Yes	Yes
C26	Is it fast&easy to setup working environment	Yes	No	Yes
C27	Is the documentation about the tool enough	Yes	Yes	No
C28	Is the community active	Yes	Yes	No

In the table 5.1.1;

- For the criteria C5, as clicking on “Home” and “Recent Apps” buttons do not work properly but clicking on “Back” button works, it is stated as “Partially” for Robotium.
- For the criteria C6, as clicking on softkeyboard button is possible but hiding softkeyboard is not, it is stated as “Partially” for UIAutomator.
- For the criteria C7, it is possible to set Wi-Fi and mobile data on/off, however, it is not possible to change mobile data settings as 2G,3G or 4G using Robotium and Appium,Thus, it is stated as “Partially” for these tools.
- For the criteria C9 as it is possible to open the notification area but not possible to click on it using UIAutomator, it is stated as “Partially” for this tool.
- For the criteria C11, as Appium supports record and replay functionality only on IOS platform, this criteria is stated as “Partially” for Appium.
- For the criteria C24, as UIAutomator allows reaching some elements of WebView, successfully, this criteria is stated as “Partially” for UIAutomator.
- For the criteria C25, Robotium requires knowledge about package name and launcher activity name of the application under test when we work with APK file. In addition, to use some Robotium functionalities such as screenshot and making connection settings, it is necessary to reach the application’s code to give permission. However, access and knowledge about the entire source code is not necessary. That is why, this criteria is stated as “Partially” for Robotium.

The table shows that, there is no single tool that meets all of the criteria. 8 criteria are met by each of the 3 tools. The criteria related to interrupt testing and leakage testing (C12-C17) is either not met by the selected tools or there is no information about them. This situation shows that, a new Android testing tool that meets these criteria may be one step forward from these popular testing tools. Record and replay property (C11) is another criteria that needs attention. Only Appium supports record and replay property and only for iOS platform. None of these three tools support recording and replaying tests on Windows.

We have also found that simulating undesired multi selection (C3) and interaction with status bar notifications (C9) are completely met only with Appium. So, if these criteria are essential for testing an application, Appium may be the most suitable tool. As Robotium and Appium cannot set the mobile data to 2G, 3G or 4G, testing the behavior of the app with different internet connection (C7) can be completely achieved only using UIAutomator. This means, if reaching these settings is an important requirement of an application’s testing process, UIAutomator may be the most suitable tool.

The table also points out that, reaching Android hardware buttons (C5), setting airplane mode (C8) and testing without knowledge about source code of the application (C25) are not completely allowed by Robotium where Appium and

UIAutomator completely allows them. So, if these criteria are needed to be met for testing process, it might not be a good idea to use Robotium. Similarly, setting up working environment (C26) is fast and easy for Robotium and UIAutomator but more difficult and time consuming for Appium. Thus, if a tool whose working environment can be prepared easily is desired, it is not a good idea to choose Appium as testing automation tool. Generating test report file (C10), a rich documentation (C27) and an active community (C28) are the criteria which are provided by Robotium and Appium but not by UIAutomator. So, it these criteria are taught as essentials, it is not a good idea to use UIAutomator. Interaction with soft keyboard (C6) and testing hybrid applications (C24) are totally enabled by Robotium and Appium where UIAutomator partially enables. Thus, if a test suite requires full interaction with the soft keyboard it is better not to use UIAutomator. Similarly, if a hybrid application is tested, UIAutomator will not be the most suitable testing tool.

We have concluded that not only for developing but also for testing Android applications, Java is the leading programming language. Java is essential to use Robotium and UIAutomator. Appium allows using any WebDriver compatible language which also includes Java.

When we examined the APIs for implementing tests for UI items in disabled mode, we have seen that, only Robotium allows some of the UI elements to disable. These elements are widgets with editview type, button type and soft keyboard. Other elements such as notification area and Android hardware buttons cannot be disabled using Robotium. On the other hand Appium and UIAutomator does not allow for disabling any UI element. This situation shows that, these 3 automation tools are lack of the functionality to disable UI elements.

We have concluded that, if a mobile application consists of a number of functions, the criteria related to functional testing (C1-C9) is more important and the tool which meets these criteria should be used for testing. If the application includes sensitive data or processes which can be affected by interrupts, the criteria related to interrupt testing (C12-C17) can be considered more significant and the tool that meets these criteria as the more suitable one. If the source code of the application under test is not reachable, the criteria related to access to source code (C20, C21, C25) can be considered more important for that application.

We have also seen that Robotium meets 14 (52%) criteria totally and 3 (11%) criteria partially. It is clearly stated that, Robotium does not meet 3 (11%) of the criteria. Robotium resources do not provide sufficient information about 7 (26%) criteria. On the other hand, Appium meets 18 (67%) criteria totally and 2 (7%) criteria partially. It is clearly stated that, Appium does not meet 2 (7%) criteria. Appium resources do not provide sufficient information about 5 (19%) criteria. UIAutomator meets 13 (48%) of the criteria totally and 3 (11%) of the criteria partially. It is stated that, UIAutomator does not meet 4 (15%) criteria. UIAutomator resources do not provide sufficient information about 7 (26%) criteria. From these statistics, we can conclude

that Appium is the tool which meets the most number of criteria and UIAutomator is the tool which meets the least number of criteria. In Figure 5.1.1, we present these statistics.

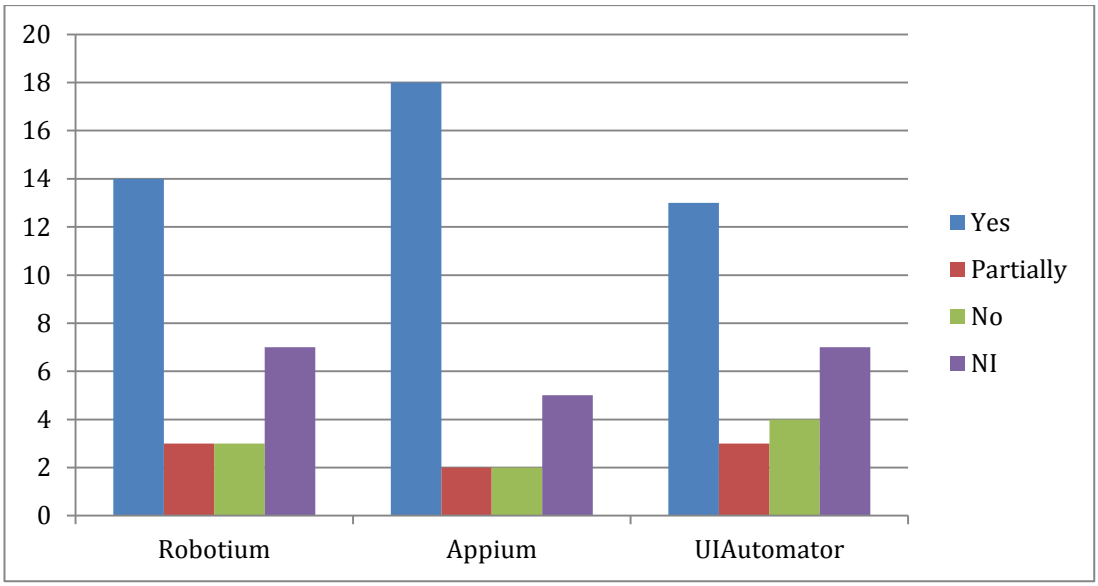

Figure 5.1.1 : The Number Of Criteria Met By Selected Tools

We have compared the three tools in terms of line of code (LOC) needed to be written for the same test suite. For the application 'DroidWeight' we have analyzed the LOC for 8 test cases which are enabled by each of the three tools. For the application '2048', we have analyzed the LOC for 5 test cases which are enabled by both Robotium and Appium. Because UIAutomator does not support hybrid application testing well, we have not analyzed its LOC for the application '2048'. Table 5.1.2 shows that, Appium is the tool that requires the most LOC for testing both native applications and hybrid applications than other tools. Robotium is the tool that requires the least LOC for testing native and hybrid applications.

Table 5.1.2: REQUIRED LOC FOR THE SAME TEST SUITE

	Robotium	**Appium**	**UIAutomator**
DroidWeight (Native App)	119	150	127
2048 (Hybrid App)	68	97	-

In order to compare execution times of the tools, we have executed the test cases which are developed to test the application 'DroidWeight' 30 times using each of the 3 tools. We have calculated the average execution time for each test case. Table 5.1.3 shows the mean execution times, standard deviations and p-values which shows the

significance of differences between execution times of Robotium, Appium and UIAutomator test cases. The table shows that, performances of the tools differentiate for different functionalities. It is not possible to select one of them as the tool that provides shortest execution time.

When we have analyzed the table in terms of significance of differences between execution times, we have gathered following results;

- For the test case 'TestInternetConnection()',as p-values are 0,000 which is lower than 0,05; the differences between mean execution times of each tool is significant. The table shows that; Robotium is significantly faster than other two tools and Appium is slower than other tools for execution of this test case.
- For the test case 'TestScreenItems()',as each p-value is lower than 0,05; the differences between mean execution times of each tool is significant. The table shows that; Robotium is significantly faster than other two tools and Appium is slower than other tools for execution of this test case.
- For the test case 'TestLongClick()', as p-values of Robotium and UIAutomator is 0,910 which is higher than 0,05; the differences between mean execution times of these tools is not significant. On the other hand, 'P-values of Appium and other two tools are lower than 0,05. This shows that, the mean execution time of Appium is significantly different from Robotium and UIAutomator. The figure shows that; Appium is significantly faster than other two tools whose execution times are nearly equal for the test case 'TestScreenItems()
- For the test case 'TestDrag()', as each p-value is 0,000 which is lower than 0,05; the differences between mean execution times of each tool is significant. The figure shows that; Appium is significantly faster than other two tools and Robotium is slower than other tools for execution of this test case.
- For the test case 'TestSwipe ()', as p- values are 0,562 which is higher than 0,05; the differences between mean execution times of Robotium and UIAutomator is not significant. On the other hand, p-values of Appium and other two tools are lower than 0,05. This shows that, the mean execution time of Appium is significantly different from Robotium and UIAutomator. The figure shows that; Appium is significantly slower than other two tools whose execution times are nearly equal for the test case 'TestSwipe()'.
- For the test case 'TestScreenShot()', as each p-value is lower than 0,05; the differences between mean execution times of each tool is significant. The figure shows that; UIAutomator is significantly faster than other two tools and Appium is slower than other tools for execution of this test case.
- For the test case 'TestSoftKeyboard()', as each p-value is 0,000 which is lower than 0,05; the differences between mean execution times of each tool is significant. The figure shows that; UIAutomator is significantly faster than

other two tools and Robotium is slower than other tools for execution of this test case.

- For the test case ‘TestHWButtons()’, as each p-value is lower than 0,05; the differences between mean execution times of each tool is significant. The figure shows that; UIAutomator is significantly faster than other two tools and Robotium is slower than other tools for execution of the test case ‘TestHWButtons()’.
- For the test case ‘TestPinch()’, as as each p-value is 0,000 which is lower than 0,05; the differences between mean execution times of each tool is significant. The figure shows that; UIAutomator is significantly faster than other two tools and Appium is slower than other tools for execution of the test case ‘TestPinch()’.

Table 5.1.3: COMPARISON AND ANALYSIS OF MEAN TEST EXECUTION TIMES

TestCase		Robotium		Appium		UIAutomator	
testInternetConnection()	**Mean**	8,496		44,461		37,804	
	Std. Deviation	1,902		3,886		1,758	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,000
		UIAutomator	0,000	UIAutomator	0,000	Appium	0,000
testScreenItems()	**Mean**	36,463		39,636		38,104	
	Std. Deviation	2,120		0,924		2,029	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,002
		UIAutomato	0,002	UIAutomato	0,004	Appium	0,004
testLongClick()	**Mean**	18,555		5,980		18,389	
	Std.	1,934		0,645		1,768	

	Deviation						
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,910
		UIAutomator	0,910	UIAutomator	0,000	Appium	0,000
testDrag()	**Mean**	32,461		13,183		22,923	
	Std. Deviation	2,754		0,793		0,625	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,000
		UIAutomato	0,000	UIAutomato	0,000	Appium	0,000
testSwipe()	**Mean**	21,694		59,818		21,124	
	Std. Deviation	2,244		2,496		1,619	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,562
		UIAutomator	0,562	UIAutomator	0,000	Appium	0,000
testScreenShot()	**Mean**	9,355		10,431		1,446	
	Std. Deviation	1,494		1,177		0,056	
	P-value	Appium	0,001	Robotium	0,001	Robotium	0,000

		UIAutomato	0,000	UIAutomato	0,000	Appium	0,000
testSoftKeyboard()	**Mean**	20,527		23,374		17,689	
	Std. Deviation	3,554		2,203		1,106	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,000
		UIAutomator	0,000	UIAutomator	0,000	Appium	0,000
testHWButtons()	**Mean**	18,379		14,883		13,974	
	Std. Deviation	1,997		0,562		1,285	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,000
		UIAutomato	0,000	UIAutomato	0,038	Appium	0,038
testPinch()	**Mean**	16,117		18,765		11,627	
	Std. Deviation	3,138		0,773		0,255	
	P-value	Appium	0,000	Robotium	0,000	Robotium	0,000
		UIAutomato	0,000	UIAutomato	0,000	Appium	0,000
testMultiTouch()	**Mean**	-		2,059		-	
testNotifications()	**Mean**	-		5,552		0,804	

5.2. LIMITATIONS

We have faced some difficulties and limitations on this study. Firstly, finding an open source Android project whose source code is easy to understand was not easy. We have searched on GitHub and FDroid to find suitable applications for our study. Some of the applications we found include errors, some of them do not work properly in our environment, and some of them have the source code which is quite complicated and hard to understand. As a result of a long search we have found 'DroidWeight' and '2048' to study.

There are limitations in our study. We have conducted our study only with open source Android testing tools. Our study does not include Android test automation tools that require subscription and payment. Another limitation is that, we worked on Windows platform only. We have not been able to test the tools' capabilities on iOS platform. Moreover, as we have studied with Android applications, we could not evaluate the cross platform testing tool; Appium on iOS applications. Thirdly, as Android testing is a new area, we have faced difficulties about reaching documentation. The number of books, official web pages and tutorials were limited. Thus, we have generally used forums and blogs to reach information we need during the study. Sometimes we could not find the information we need on any of the documents we reached. When we faced this situation, we have created topics on forums and opened issues on GitHub. In addition, we have focused on GUI testing, interrupt testing and leakage testing. The automation tools have not evaluated with respect to other testing types. Another limitation of our study is that, we have conducted our experimental study on medium sized applications. If the test suites are executed on complex applications, some of the test cases may fail, so; the results may not be stable. Finally, we have executed all test suites on the same environment without any changes on hardware and operating system. If the setup environment changes, execution times may change for each tool and each test case.

5.3. CONCLUSION AND FUTURE WORK

In this thesis study, we have compared three most popular open source Android test automation tools. In order to achieve this, firstly we have identified the criteria which is used for comparison of the tools. We have examined blogs, checklists and related studies and conducted an exploratory study for criteria identification. We have aimed to make a list of criteria which is not only usable for our study but also for future studies related to mobile application testing and mobile application testing tools. Then, we have evaluated the tools with respect to these criteria. In order to accomplish this, we have conducted a research and a case study. In this case study, we have created test suites for 2 Android applications using each of the selected automation tools. According to the research and case study, we have compared these tools in terms of the criteria they meet, required line of code and test execution times.

As a result of our study, we have concluded that, each automation tool has some strong and weak points. There is not a single tool that meets all of the criteria. Thus, it is not possible to point out one tool as the best or the most advantageous one. The most suitable test automation tool for an Android application depends on the requirements and properties of the application.

We also observed that there is a need for test automation for simulating incoming call, sms, push notification, alarm, simulating effects of inserting and removing charger and simulating lack of battery and memory.

As future work, there may be some improvements on this study. We plan to add IOS and more cross platform test automation tools to this comparative study. We also plan to enhance the comparison criteria by adding new ones and update existing ones as more detailed. We plan to prioritize the criteria by giving coefficients for each. Further studies could be also conducted on the comparison of mobile testing tools which are not free and open source.

REFERENCES

1. Smartphone. (n.d.). Retrieved March 04, 2016, from https://en.wikipedia.org/wiki/Smartphone
2. The Statistics Portal: Number of available applications in the Google Play Store from December 2009 to February 2016.(n.d.). Retrieved March 05, 2016, from http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
3. Gao, J., Bai,X., Tsai, W. & Uehara,T. (2014). Mobile Application Testing:A Tutorial. Computer Magazine. 47 (2). 46-55
4. Test automation. (n.d.). Retrieved March 05, 2016, from https://en.wikipedia.org/wiki/Test_automation
5. Mobile application testing. (n.d.). Retrieved March 05, 2016, from https://en.wikipedia.org/wiki/Mobile_application_testing
6. Gunasekaran,S. & Bargavi,V. (2015) Survey on Automation Testing Tools For Mobile Applications. International Journal of Advanced Engineering Research and Science (IJAERS). 2(11). 36-41
7. Singh,S., Gadgil,R. & Chudgor,A. (2014). Automated Testing of Mobile Applications using Scripting Tecnnique: A Study on Appium. International Journal of Current Engineering and Technology (IJCET). 4(5). 3627-3630
8. Tools Help. (n.d.). Retrieved March 05, 2016, from http://developer.android.com/tools/help/index.html
9. Application Fundamentals. (n.d.). Retrieved March 05, 2016, from http://developer.android.com/guide/components/fundamentals.html
10. Running Apps in the Android Emulator. (n.d.). Retrieved March 05, 2016, from http://developer.android.com/tools/devices/emulator.html
11. Android Testing Tools. (n.d.). Retrieved March 07, 2016, from http://developer.android.com/tools/testing/testing-tools.html
12. Muccini,H., Francesco,A. & Esposito,P. (2012). Software Testing of Mobile Applications: Challenges and Future Research Directions, presented at 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland, 2012. IEEE.
13. Testing Support Library. (n.d.). Retrieved March 08, 2016, from http://developer.android.com/tools/testing-support-library/index.html
14. AndroidJunitRunner. (n.d.). Retrieved March 08, 2016 from http://developer.android.com/reference/android/support/test/runner/AndroidJUnitRunner.html
15. Robotium:FAQ. (n.d.). Retrieved March 07, 2016 from http://robotium.com/pages/faq
16. Zadgaonkar, H. (2013). Robotium Automated Testing for Android. Retrieved from https://ebooks-it.org/178216801x-ebook.html

17. GitHub: RobotiumTech/Robotium. (n.d.). Retrieved January 15, 2016 from https://github.com/RobotiumTech/robotium
18. The History of Appium. (n.d.). Retrieved February 07, 2016 from http://appium.io/history.html?lang=tr
19. GitHub: appium/Appium. (n.d.). Retrieved January 25, 2016 from https://github.com/appium/appium
20. Appium Design. (n.d.). Retrieved March 02, 2016 from http://appium.io/introduction.html
21. Bayley,I., Flood,D., Harrison,R. & Martin,C. (2012). MobiTest: A Cross-Platform Tool for Testing Mobile Applications, presented at 7th International Conference on Software Engineering Advances (ICSEA), Lisbon, Portugal, 2012. IARIA.
22. Shah,G., Shah,P. & Muchhala,R. (2014). Software Testing Automation using Appium. International Journal of Current Engineering and Technology (IJCET). 4(5). 3528-3531
23. Jain,A., Jain,M. & Dhankar,S. (2014). A Comparison of RANOREX and QTP Automated Testing Tools and their impact on Software Testing. International Journal of Engineering, Management & Sciences (IJEMS). 1(1). 8-12
24. Kaur,H. & Gupta,G. (2013). Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete. International Journal of Engineering Research and Application (IJERA). 3(5). 1739-1743
25. Dalmasso,I., Datta,K.S., Bonnet,C & Nikaein, N. (2013). Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools, presented at 9th International Wireless Communications and Mobile Computing Conference (IWCMC). Sardinia, Italy, 2013. IEEE.
26. Rani,S., Suri,B, & Khatri,S.K. (2015). Experimental Comparison of Automated Mutation Testing Tools for Java, presented at 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) , Noida,India, 2015. IEEE.
27. Kumar,R. & Singh, A.J. (2015). A Comparative Study and Analysis of Web Service Testing Tools. A Monthly Journal of Computer Science and Information Technology (IJCSMC). 4(1). 433-442
28. Sauce Labs. (n.d.). Retrieved March 1, 2016 https://en.wikipedia.org/wiki/Sauce_Labs
29. Mobile Testing Tools - 11 Open Source Frameworks Compared. (n.d.). Retrieved January 8, 2015 from https://saucelabs.com/resources/mobile-testing-tools
30. Testdroid. (n.d.). Retrieved March 1, 2016 from https://en.wikipedia.org/wiki/Testdroid
31. Shao,L. (2015). Top 5 Android Testing Frameworks (with Examples). Retrieved from http://testdroid.com/tech/top-5-android-testing-frameworks-with-examples

32. Test Management Approach. (n.d.). Retrieved January 8, 2016 from https://en.wikipedia.org/wiki/Test_Management_Approach
33. Checklist Mobile App Testing. (n.d.). Retrieved January 8, 2016 from http://www.tmap.net/downloads
34. Suvesh,T.K. & Sanoj,S. (2014). How to Evaluate a Mobile Test Automation Tools for your Application?. Retrieved from http://www.rapidvaluesolutions.com/how-to-evaluate-a-mobile-test-automation-tool-for-your-application/
35. Khode,A. (2012). Testing Checklist for Mobile Applications. Retrieved from http://www.mobileappstesting.com/testing-checklist-for-mobile-applications/
36. Top 10 Mobile Testing Tools. (2015). Retrieved from http://www.optimusinfo.com/blog/top-10-mobile-testing-tools/
37. Ghahrai, A. (2014). 10+ Open Source Mobile Test Automation Tools. Retrieved from http://www.testingexcellence.com/open-source-mobile-test-automation-tools/
38. Baluk,M. & Miscellaneous. (2014). 5 Open Source Tools for Android App Test Automation. Retrieved from http://www.testlab4apps.com/5-open-source-tools-for-android-app-test-automation/
39. Android calculator application. (n.d.). Retrieved January 3, 2016 from https://sourceforge.net/projects/androidcalculat/
40. Droid Weight – Android Application in Google Play Store. (n.d.). Retrieved January 15, 2016 from https://play.google.com/store/apps/details?id=de.delusions.measure&hl=tr
41. GitHub: aymanstar/droidweight. (n.d.). Retrieved January 15, 2016 from https://github.com/aymanstar/droidweight
42. GitHub: uberspot/2048. (n.d.). Retrieved January 22, 2016 from https://github.com/uberspot/2048-android
43. Gurram,S. (2012). What is Robotium. Retrieved from http://robotiumsolo.blogspot.com.tr/2012/12/what-is-robotium.html
44. Plotytsia, S. (2014). How to Choose the Right Mobile Test Automation Tool?. Retrieved from http://www.testlab4apps.com/how-to-choose-the-right-mobile-test-automation-tool/
45. Bristove,J. (2015). What is a Hybrid Mobile App?. Retrieved from http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/
46. Gestures. (n.d.). Retrieved February 15, 2016 from https://www.google.com/design/spec/patterns/gestures.html#gestures-drag-swipe-or-fling-details
47. com.jayway.android.robotium.solo.Solo. (n.d.). Retrieved January 16, 2016 from http://grepcode.com/file/repo1.maven.org/maven2/com.jayway.android.robotium/robotium-solo/4.2/com/jayway/android/robotium/solo/Solo.java

48. Joshi,S. (2014). 15+ Useful Robotium Code Snippets for Android Test Automation. Retrieved from https://www.javacodegeeks.com/2014/06/15-useful-robotium-code-snippets-for-android-test-automation.html
49. Key Event. (n.d.). Retrieved January 20, 2016 from http://developer.android.com/reference/android/view/KeyEvent.html
50. Class SystemUtils. (n.d.). Retrieved January 20, 2016 from https://robotium.googlecode.com/svn/doc/com/robotium/solo/SystemUtils.html
51. Robotium FAQ. (n.d.). Retrieved January 3, 2016 from http://robotium.com/pages/faq
52. Robotium Developers Google Group- Simulating incoming call using Robotium. (n.d.). Retrieved January 22, 2016 from https://groups.google.com/forum/#!topic/robotium-developers/Za8B-6GS9Ps
53. Black Box Testing With Robotium On APK Files. (n.d.). Retrieved January 25, 2016 from https://robotium.googlecode.com/files/RobotiumForBeginners.pdf
54. WebView. (n.d.). Retrieved March 22, 2016 from http://developer.android.com/reference/android/webkit/WebView.html
55. Robotium Developers Google Group. (n.d.). Retrieved April 3, 2016 from https://groups.google.com/forum/#!forum/robotium-developers
56. Stackoverflow: Tags. (n.d.). Retrieved April 3, 2016 from http://stackoverflow.com/tags
57. Appium Tutorials. (n.d.). Retrieved February 2, 2016 from http://software-testing-tutorials-automation.blogspot.com.tr/2015/10/appium-tutorials.html
58. Hans,M. (2015). Advanced User Interactions. Packt Publishing (ed.), Appium Essentials. Retrieved from https://www.safaribooksonline.com/library/view/appium-essentials/9781784392482/ch07.html
59. TestNG. (n.d.). Retrieved February 11, 2016 from http://testng.org/doc/index.html
60. Appium Tutorial for Beginners. (n.d.). Retrieved February 5, 2016 from http://www.guru99.com/introduction-to-appium.html
61. Appium Android Setup. (2015). Retrieved from http://www.seleniumtests.com/2015/05/appium-and-android-setup.html
62. Appium Top 50 Real Time Interview Questions Evergreen. (2015). Retrieved from https://www.linkedin.com/pulse/appium-top-50-real-time-interview-questions-evergreen-akhil-reddy
63. GitHub: Appium Issues. (2016). Retrieved from https://github.com/appium/appium/issues/6179
64. What is Appium? Why Need Appium? Limitations of Appium. (n.d.). Retrieved February 20, 2016 from http://software-testing-tutorials-automation.blogspot.com.tr/2015/09/what-is-appium-why-need-appium.html

65. Appium: Getting Started. (n.d.). Retrieved March 02, 2016 from http://appium.io/getting-started.html
66. Stackoverflow: Questions – Run Appium Test on Command Line. (2016). Retrieved from http://stackoverflow.com/questions/35738517/run-appium-test-on-command-line/35738746#35738746
67. TestNG. (n.d.). TestNG Ant Task. Retrieved March 12, 2016 from http://testng.org/doc/ant.html
68. Creating And Running WebDriver Test Suit Using testng.xml file.(n.d.). Retrieved March 15, 2016 from http://software-testing-tutorials-automation.blogspot.com.tr/2014/03/creating-and-running-test-suit-using.html
69. Appium Discussion Group. (n.d.). Retrieved January 2, 2016 from https://discuss.appium.io/
70. UiObject Class (n.d.). Retrieved February 21, 2016 from http://developer.android.com/reference/android/support/test/uiautomator/UiObject.html
71. Stackoverflow: Questions- How to Achieve Long Click in UiAutomator. (2014). Retrieved from http://stackoverflow.com/questions/21432561/how-to-achieve-long-click-in-uiautomator
72. Stackoverflow: Questions – UIAutomator Test Report. (2016). Retrieved from http://stackoverflow.com/questions/36240732/uiautomator-test-report/36255451#36255451
73. Patil,K. (2015). Top 5 Open Source Automation Tools For Ios And Android (Infographic). Retrieved from http://afourtech.com/automation-tools-for-ios-and-android-apps/
74. Testing UI For Multiple Apps. (n.d.). Retrieved March 12, 2016 from http://developer.android.com/training/testing/ui-testing/uiautomator-testing.html
75. Uiautomator: Starting with uiautomator. (2013). Retrieved from http://uiautomatortester.blogspot.com.tr/2013/09/starting-with-uiautomator.html
76. Chakraborty, J. (2015). Top 5 Android Testing Frameworks. Retrieved from https://www.linkedin.com/pulse/top-5-android-testing-frameworks-jaybrata-chakraborty?trk=prof-post&trkSplashRedir=true&forceNoSplash=true
77. Helppi, V.V. (2013). The Pros and Cons of Different Android Testing Methods. Retrieved from http://testdroid.com/news/the-pros-and-cons-of-different-android-testing-methods
78. GitHub: uiautomator issues. (n.d.). Retrieved March 3, 2016 from https://github.com/xiaocong/uiautomator/issues
79. Android Studio The Official IDE for Android. (n.d.). Retrieved December 15, 2015 from http://developer.android.com/sdk/index.html
80. GitHub: RobotiumTech/Robotium. (n.d.). Retrieved January 2, 2016 from https://github.com/RobotiumTech/robotium/wiki/Downloads

81. Appium Download. (n.d.). Retrieved January 10, 2016 from https://bitbucket.org/appium/appium.app/downloads/
82. What is Equivalence partitioning in Software testing?. (n.d.). Retrieved March 14, 2016 from http://istqbexamcertification.com/what-is-equivalence-partitioning-in-software-testing/
83. Unix Time. (n.d.). Retrieved March 10, 2016 from https://en.wikipedia.org/wiki/Unix_time
84. App Manifest. (n.d.). Retrieved April 13, 2016 from http://developer.android.com/guide/topics/manifest/manifest-intro.html