A TEST CASE GENERATION APPROACH FOR MOBILE APPS BASED ON CONTEXT AND GUI EVENTS

ASMAU USMAN

A thesis submitted in
fulfilment of the requirement for the award of the
Degree of Master of Information Technology

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

AUGUST 2018
DEDICATION

To my beloved mother and my late father.
ACKNOWLEDGEMENT

My heartiest gratitude to Allah (SWT) for making this journey a successful one. Firstly, I would like to express my sincere appreciation to my supervisor Dr. Noraini Ibrahim, for her support, guidance and criticism during my study.

Very special thanks to my husband Dr. Ibrahim Salihu Anka for his never-ending support, encouragement and understanding through my good and bad times. I would also like to express my appreciation and gratitude to my family especially my Mother, Mother in-law, Brothers and Sisters for their support and prayers towards my success. Same goes to my Children Zulaikha, Yusuf, Aliyu and Aysha for their support, and patient towards the success of this thesis.

My special appreciation goes to the members of staff at the Faculty of Computer Science and Information Technology (FSKTM), and the Centre for Graduate Studies, Universiti Tun Hussein Onn Malaysia (UTHM) for providing a conducive learning environment for students. I would like to acknowledge the support from UTHM in undertaking the research under the Graduate Research Assistant for Postgraduate Research Grants (GPPS), Universiti Tun Hussein Onn Malaysia and Ministry of Higher Education through Fundamental Research Grant Scheme (FRGS) vote number 1610. I would like to express my appreciation to my friend Farida Dikko (Mrs Aliyu Kurfi) for her support and kindness. I would also like to thank my research colleagues at the faculty and friends for their insightful discussions, collaborations and support.
ABSTRACT

The increase of mobile devices with rich innovative feature has become an enabler for developing mobile applications (mobile apps) that offer users an advance and extremely-localized context-aware content. Nowadays mobile apps are developed to address more critical areas of people’s daily computing needs, which bring concern on the applications’ quality. In order to build a high quality and more reliable applications, there is a need for effective testing techniques to test the apps. The most recent testing technique focuses on graphical user interface (GUI) events with little attention to context events. This makes it difficult to identify other defects in the changes that can be inclined by context in which an application runs. The major challenge in testing mobile apps that react to context events is how to identify the events from an application during testing. This study proposes an approach (named TEGDroid) for testing mobile apps considering the two sets of events: GUI and context events. This approach comprises five steps which are; extraction of resources from APK file, static analysis of the extracted app’s byte code to identify GUI events, analysis of mobile apps’ permission to identify different scenarios of context events, generation of test case based on the GUI and context events and validation of the test cases using code coverage and mutation testing. Experiment was performed on real world open source mobile apps to evaluate TEGDroid. Results from the experimental evaluation indicates that the approach is effective in identifying context events and had 61%-91% coverage across the seven (7) selected applications. Results from the mutation analysis shows that 100% of the mutants were killed. This indicates that TEGDroid have the capability to detect faults in mobile apps.
ABSTRAK

Peningkatan peranti mudah alih yang kaya dengan ciri-ciri inovatif telah menyumbang kepada pembangunan aplikasi mudah alih yang menawarkan pengguna dengan kandungan yang terbaru dan disesuaikan dengan konteks. Kini aplikasi mudah alih dibangunkan untuk menangani lebih banyak bidang kritikal dalam keperluan komputeran harian manusia, yang membawa kepada keprihatinan terhadap kualiti aplikasi. Kebanyakan teknik ujian terkini memfokuskan kepada peristiwa GUI dan kurang perhatian kepada peristiwa konteks. Ini menjadikan ianya sulit untuk mengenal pasti keadaan keadaan pada keadaan yang berubah-ubah mengikut keadaan konteks perjalanan aplikasi. Cabaran utama dalam menguji aplikasi mudah alih yang bertindak balas terhadap peristiwa konteks ialah bagaimana mengenal pasti peristiwa-peristiwa tersebut semasa pengujian aplikasi. Kajian ini mencadangkan satu pendekatan yang dinamakan TEGDroid untuk menguji aplikasi mudah alih yang menimbangkan kedua-dua set peristiwa iaitu peristiwa GUI dan konteks. Pendekatan yang dicadangkan terdiri daripada lima langkah iaitu; mengekstrak sumber dari fail APK, melakukan static analisis terhadap bait kod aplikasi yang telah diekstrak untuk mengenal pasti peristiwa GUI, menganalisis fail izin aplikasi mudah alih untuk mengenal pasti senario yang berbeza daripada peristiwa konteks, menjana kes ujian berdasarkan peristiwa GUI dan konteks dan mengesahkan kes ujian menggunakan kaedah liputan kod dan ujian mutasi. Satu eksperimen telah dilaksanakan menggunakan beberapa sumber terbuka aplikasi mudah alih yang sebenar untuk menilai TEGDroid. Hasil eksperimen menunjukkan bahawa pendekatan ini berkesan dalam mengenal pasti peristiwa konteks dan mempunyai liputan sebanyak 61% -91% di semua tujuh (7) aplikasi terpilih. Untuk menilai keupayaan pendekatan ini mengesahkan kesalahan, ujian mutasi dilakukan dengan memperkenalkan mutan kepada aplikasi. Hasil dari analisis mutasi menunjukkan bahawa 100% mutan terbunuh. Ini menunjukkan bahawa TEGDroid mempunyai keupayaan untuk mengesahkan kesalahan dalam aplikasi mudah alih.
CONTENTS

TITLE
DECLARATION
DEDICATION
ACKNOWLEDGEMENT
ABSTRACT
ABSTRAK
TABLE OF CONTENTS
LIST OF TABLES
LIST OF FIGURES
LIST OF ALGORITHMS
LIST OF ABBREVIATIONS
LIST OF APPENDICES
LIST OF PUBLICATIONS

CHAPTER 1 INTRODUCTION
  1.1 Problem Statement
  1.2 Research Objectives
  1.3 Scope
  1.4 Significance of Research
  1.5 Thesis Organization

CHAPTER 2 LITERATURE REVIEW
  2.1 Android Platform Architecture
  2.2 Android Mobile Application
    2.2.1 Mobile App Components
    2.2.2 Application Package File (APK)
    2.2.3 Mobile App Events
    2.2.4 Mobile App’s Permissions
  2.3 Mobile App Testing
    2.3.1 Manual Testing
CHAPTER 3 RESEARCH METHODOLOGY

3.1 Research Process
   3.1.1 Literature Review
   3.1.2 Methodology
   3.1.3 Implementation of Proposed Approach
   3.1.4 Validation
       3.1.4.1 Experimental Setup
3.2 Research Framework
3.3 Step 1: Resources Extraction
   3.3.1 Decompiling DEX
   3.3.2 De-coding AndroidManifest.xml
3.4 Step 2: Static Analysis
3.5 Step 3: Permission Extraction
   3.5.1 AndroidManifest.xml Analysis
   3.5.2 Resource Combination
3.6 Step 4: Test Case Generation
3.7 Step 5: Validation
   3.7.1 Code Coverage
   3.7.2 Fault Detection using Mutation Testing
3.8 Dataset and Experimental Setup
   3.8.1 Dataset
   3.8.2 Experimental Setup
CHAPTER 4 IMPLEMENTATION AND EXPERIMENTATION 46

4.1 Case Study 46

4.1.1 Overview of Beem 46

4.2 Step 1: Resource Extraction 47

4.2.1 Decompiling DEX 48

4.2.2 De-coding AndroidManifest.xml 49

4.3 Step 2: Static Analysis 49

4.3.1 Events Generation 52

4.4 Step 3: Permission Extraction 55

4.4.1 Resource Combination 56

4.5 Step 4: Test Case Generation 57

4.6 Chapter Summary 61

CHAPTER 5 RESULTS AND DISCUSSIONS 62

5.1 Experimentation 62

5.2 Coverage Results 63

5.3 Comparison of Coverage Results 64

5.3.1 Comparison of TEGDroid with Extended Ripper 64

5.3.2 Comparison of TEGDroid with Dynodroid 66

5.3.3 Comparison of TEGDroid with Song et al. 67

5.3.4 Comparison of TEGDroid with EHBDroid 68

5.4 Fault Detection 69

5.5 Chapter Summary 72

CHAPTER 6 CONCLUSION AND FUTURE WORK 73

6.1 Achievements of Objectives 73

6.2 Contribution of the Study 75

6.3 Recommendation for Future Work 75

REFERENCES 77

APPENDICES 88
LIST OF TABLES

2.1 Comparison of Methods of Events Identification of Approach 26
2.2 Comparison of testing techniques used by approach 27
3.1 Mutation Operators for muDroid 43
3.2 Summary of Datasets for the Experiment 44
4.1 List of Permissions and their Resources 47
4.2 List of Permissions with their Related Resources 56
4.3 Results of Resource Combination 57
4.4 Number of Test Case Generated 61
5.1 Mobile Apps used in Evaluation 63
5.2 Code Coverage Result 63
5.3 Comparison of TEGDroid with Extended Ripper 64
5.4 Comparison of TEGDroid with Dynodroid 66
5.5 Comparison of TEGDroid with Song et al. 67
5.6 Comparison of TEGDroid with EHBdroid 68
5.7 Mutants Generated for Seven Selected Apps 69
5.8 Results of Mutation Testing. 71
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Android Framework Architecture</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Number of Mobile Applications in Google Play Store</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Generated Revenue of Mobile Apps</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>The structure of Application Package (APK)</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Comparison Criteria</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Research Process</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Research Framework</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Resource Extraction from APK File</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Snapshot of Extracted Resources from Beem App</td>
<td>48</td>
</tr>
<tr>
<td>4.2</td>
<td>Manifest.xml file Extracted from Beem App</td>
<td>49</td>
</tr>
<tr>
<td>4.3</td>
<td>Snapshot of Beem code showing an example callback method</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of Windows Transition Graph for Beem App</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Screenshot for app’s events generation from Beem</td>
<td>54</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of an event from the $eS$ of Beem app</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>An example of Test Case Generated for Beem App</td>
<td>58</td>
</tr>
<tr>
<td>4.8</td>
<td>An example of Test Case with Context Event Generated from Beem App</td>
<td>59</td>
</tr>
<tr>
<td>4.9</td>
<td>Test Case for Context Event Generated from Resources States</td>
<td>60</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparison of Coverage Result with Extended Ripper</td>
<td>65</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparison of Coverage Result with Dynodroid</td>
<td>66</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparison of Coverage Result with Song et al.</td>
<td>67</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Comparison of Coverage Result with EHBdroid</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Percentage of Generated and Selected Mutants</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>Total Number of Mutants for each Mutation Operator</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>Mutation Score of Operators</td>
<td></td>
</tr>
</tbody>
</table>
## LIST OF ALGORITHMS

3.1 Static Analysis  
3.2 Permission Extraction  
3.3 Generate Test Case

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Static Analysis</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>Permission Extraction</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Generate Test Case</td>
<td>40</td>
</tr>
</tbody>
</table>
LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APK</td>
<td>Application Package File</td>
</tr>
<tr>
<td>ART</td>
<td>Android Run Time</td>
</tr>
<tr>
<td>AOR</td>
<td>Arithmetic Operator Replacement</td>
</tr>
<tr>
<td>C</td>
<td>Coverage</td>
</tr>
<tr>
<td>DVM</td>
<td>Dalvik Virtual Machine</td>
</tr>
<tr>
<td>EDS</td>
<td>Event Driven Software</td>
</tr>
<tr>
<td>Eh</td>
<td>Event Handler</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning system</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>GW</td>
<td>GUI Widget</td>
</tr>
<tr>
<td>HLL</td>
<td>High Level Language</td>
</tr>
<tr>
<td>ICR</td>
<td>Incline Constant Replacement</td>
</tr>
<tr>
<td>ID</td>
<td>Event id</td>
</tr>
<tr>
<td>LCR</td>
<td>Logical Connector Replacement</td>
</tr>
<tr>
<td>LOC</td>
<td>Line of Code</td>
</tr>
<tr>
<td>MK</td>
<td>Mutant Killed</td>
</tr>
<tr>
<td>MS</td>
<td>Mutation Score</td>
</tr>
<tr>
<td>NOI</td>
<td>Negative Operator Inversion</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>ROR</td>
<td>Relational Operator Replacement</td>
</tr>
<tr>
<td>RVR</td>
<td>Return Value Replacement</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SUT</td>
<td>Software Under Test</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>TM</td>
<td>Total Mutant</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>W</td>
<td>Windows</td>
</tr>
<tr>
<td>WTG</td>
<td>Windows Transition Language</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Mark-up Language</td>
</tr>
<tr>
<td>TEGDroid</td>
<td>Test case generation approach for android apps</td>
</tr>
</tbody>
</table>
# LIST OF APPENDICES

<table>
<thead>
<tr>
<th>APPENDIX</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Events Summaries</td>
<td>88</td>
</tr>
<tr>
<td>B</td>
<td>Test Cases</td>
<td>98</td>
</tr>
</tbody>
</table>
LIST OF PUBLICATIONS

Journal:


Proceeding:

CHAPTER 1

INTRODUCTION

Smartphones are becoming increasingly popular in recent years. Smartphones applications (otherwise known as mobile apps) are now widely used by many people for several computational tasks due to their increase in functionality and compatibility. There are several vendors offering different smartphones built on different platforms and features. The popular platforms of smartphones used worldwide are iOS by Apple, Android by Google, and Windows by BlackBerry limited. Android has become the most popular operating system for several mobile devices including smartphones [1].

The increase of mobile devices with rich innovative features that support mobility and access to phone hardware such as sensors, camera, cellular networks and Wi-Fi have become an enabler for developing mobile apps that offer users an advance and extremely-localized context-aware content. Mobile apps are now equipped with the ability to recognize their computing context and to adapt and react to actions from the context. This category of mobile apps belongs to the context-aware computing [2]. Context can be defined as any information that can be used to characterize the situation of an entity. An entity is referred to anything that is considered relevant to the interaction between a user and an application such as place, person, and object including user and applications themselves [2]. Therefore, the behavior of a mobile app can be affected by the situation where a smartphone exists that is the computing context.

In an effort to improve reliability and quality of mobile apps and to gain recognition in the high-level competitive app’s market, developers are now resorting to software testing to improve users’ confidence on their applications [3-6]. Furthermore, people’s dependence on mobile apps for various computational needs poses a significant concern on the quality of apps.
Software testing is one of the important stages in software application development lifecycle that can play a significant role in improving the quality of software system [3, 5, 7]. Software testing is defined as a process, or a series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended [8]. It involves investigating the software to ascertain the quality of the product under test [9]. The primary aim of this process is to detect and prevent defects as well as to ensure the intended behaviour of the tested software [10, 11]. Software testing can be a difficult task because of the vast array of programming languages, operating systems and hardware platforms that have evolved [8]. Though the area of software testing has been growing rapidly in recent years, nonetheless, the domain of mobile app testing is still in its infancy. As a result, there is a need for more contribution in terms of ideas, approaches and techniques in the area of mobile app testing [12].

During testing, test cases can be generated manually from some information or from some software artefacts automatically. Test cases must be generated from some information, specifically some software artefacts. The software artefacts that is used includes: software specification documents; software program; information about input/output and information dynamically obtain from program execution [13]. With the recent development of mobile apps, manual testing can no longer be sufficient, because it is often tedious, error-prone and insufficient to achieve high coverage [14, 15].

Test automation is becoming increasingly popular among the software engineering community in recent times [16, 17]. Most of the approaches dedicated to mobile apps dynamically analyse an application to generate a sequence of events that can later be used as test cases to test an application. Several approaches and techniques for test automation were proposed in the past few years. Example of such techniques are GUIRipper [18], A3E [19], PUMA [20], TEMA [21], ExtendedRipper [22], AMOGA [23]. However, existing techniques for automated testing of Android apps have some limitations in dealing with context events. Therefore, this study aims to propose an automated testing approach that considers GUI and context events supported by mobile apps.
1.1 Problem Statement

In recent years mobile apps are developed to address more critical areas of people’s daily computing needs, which brought concern on the quality of applications. In order to build high quality and more reliable applications that can gain recognition in the high-level competitive application’s market, there is a need for effective testing techniques to validate the quality of the applications. The techniques should be able to validate different types of events supported by mobile apps [24] in order to improve users’ confidence in the mobile apps [3-5]. Testing event-driven applications present a great challenge to software testers such as the need to generate a huge number of possible event sequences that could sufficiently cover the application’s state space [25, 26].

In conventional software testing, test cases are constructed by exploring an application's functionality and the sequences of interactions by the user. However, for most mobile apps, there can be other events triggered from external sources other than the user interactions, such as changes in context (e.g. changing network availability, relocation, change in availability of sensor data), which may occur at any time and may have an impact on the application’s behaviour [25, 27]. These changes can expand the set of test cases for testing a mobile app significantly [6, 28].

Numerous testing techniques have been proposed for testing mobile apps in the past few years. However, most of the testing techniques for mobile apps generate test cases considering only GUI events such as [3, 5, 18, 29-33] without sufficient support for testing context events [27, 34]. Therefore, it will be difficult to identify other defects in the changes in the contexts, which can be inclined by the context in which an application runs [34]. In order to ensure that these applications behave correctly, external context events must be considered during testing such as those from GPS location data, sensors, network in addition to the GUI events.

Testing mobile apps context events have numerous challenges. The major challenge is how to identify the context events from an application during testing [22, 35]. Currently there are few testing approaches and techniques that addressed testing context events for mobile apps such as [22, 34, 36, 37]. The current approaches such as [22] dealt with the challenge of identifying context events by analysing bugs report creating an event pattern which is used to generate a sequence of context events and
use it for test generation. While the approach in [34] dealt with the problem by analysing application’s permissions to identify permissions used by an app with the corresponding resources. Then permutation is applied to the extracted permission and their resources to generate sequence of context events that can run on an app. With these approaches, the precise event sequences are generated considering this restricted number of scenarios. In view of this, the events that may trigger a faulty behaviour in an app may not be identified accurately.

Dynodroid [37] uses the Adaptive Random technique (ART) to generate a sequence of events that can be used to systematically explore an application. One of the limitations of the approach is the restriction of the apps under test from communicating with other apps. As many Android apps communicate with other apps for shared functionality and some context could not be detected. On the other hand EHBDroid [36] uses static analysis on the bytecode that invoke callback of event handlers randomly to explore the application. The static analysis is complemented with XML parsing that analyse the Androidmanifest.xml resource (permissions) file. However, it is only able to analyse limited number of callbacks and the callbacks are invoke randomly which will lead to missing some events, because some states depend on other states to happen.

Thus, the problem to be addressed in this research are the identification of context events supported by an application and executing sequence of the events. Hence, a test case generation approach called TEGDroid is proposed. TEGDroid performed static analysis on the app’s bytecode which tracks the callbacks and Intent messaging system to identify the events in sequences. It is further complemented by the analysis of the Androidmainefest.xml file to identify app’s permissions and the resources related to the permissions. The information is used to generate additional states of the context events in order to improve coverage of different scenarios of events.

1.2 Research Objectives

The aim of this study is to propose an approach for generating test case for mobile apps considering both GUI and context events. To achieve this aim, three objectives were derived as follows:
i. To propose an enhanced approach for identifying both context and GUI events from mobile applications through the static analysis of application’s manifest file and bytecode.

ii. To implement the proposed approach for test case generation for mobile applications.

iii. To validate the proposed approach by measuring the code coverage and fault detection capability using mutation testing technique.

1.3 Scope

The research focuses on implementing the proposed approach to identify context and GUI events from a mobile app and applying the proposed approach to test open source mobile apps as the test data. Seven open source mobile apps that were used in validating other approaches were selected for the test. They are barcodeScanner, beem, openCamera, pedometer, marine compass, subsonicMusicStreamer and TippyTipper. They are downloaded from GitHub [38] and SourceForge [39] databases.

1.4 Significance of Research

With the over reliance on mobile devices and use of mobile apps in safety and critical domains, the quality of mobile apps should not be compromised [3, 5]. Therefore, it is essential to develop techniques that can aid the testing of mobile apps [14, 40]. In view of this, the research is significant to developers and testers by aiding test case generation for testing mobile apps with the aim of improving the quality and performance of their apps.
1.5 Thesis Organization

The thesis comprises six chapters which include Introduction, Literature Review, Research Methodology, Implementation, Result and Discussion and finally Conclusion. The outline of each chapter is as follows.

Chapter 1 presents the background of the study, statement of problem, objectives and scope of the study and also discussed the significance of the study.

Chapter 2 presents a discussion on Android architecture and its mobile app. The structure of Android mobile apps and events supported by the apps was discussed in the chapter. The chapter further discusses software testing and test automation and stages. The chapter concludes with a review of test automation approaches and techniques for mobile apps and gives a comparative analysis to highlight limitations of the current techniques.

Chapter 3 discusses details of the methodology employed in carrying out the research. The chapter further described the experimental setup and how the proposed approach was validated.

Chapter 4 describes the implementation of the proposed approach that involves static analysis of app’s bytecode and Androidmanifest.xml file. The chapter further presents the application of the proposed approach on Beem app to generate test cases for the app. Validation of the proposed approach which involves measuring the code coverage achieved and evaluating the fault detection ability is presented in the chapter.

Chapter 5 presents the results of the experimental evaluation and discussion on the results. It concludes with a comparison of the proposed approach with other state-of-the-art approaches.

Chapter 6 concludes the thesis with the research achievements, contributions and some recommendations were also provided for further future research.
CHAPTER 2

LITERATURE REVIEW

This chapter presents an overview of Android platform and its mobile apps to give the readers a broader understanding of issues in Android mobile apps. The chapter further discusses software testing and review existing testing techniques with their features. It concludes by reviewing current approaches for testing mobile apps and presenting a comparative analysis of the approaches to highlight their strengths and weaknesses.

2.1 Android Platform Architecture

Android is an open source software platform for mobile devices built on Linux Kernel. Initially, Android was created by Android Inc., a California-based company that worked with operating systems for digital cameras and mobiles which was later acquired by Google Inc. in 2005 [41]. The Android operating system is based on Dalvik virtual machine for executing programs written in Java which compiles the code to .dex format [42]. Hence, they differ from standard Java client-server applications and traditional event-based desktop applications. Everyone can download and modify Android, but the official release needs to be approved by Google, this gives profit to the manufacturers as the availability of Android operating system-based devices increases enormously. It comprises a stack of software components structured in layered architecture shown in Figure 2.1.
The Android software stack is divided into six sections and five major layers [43] namely:

- **Application layer.** This is the topmost layer of the Android architecture and where applications are installed. By default, it contains some default applications such as Contacts Books, Google Maps, Browser, and Games. The end user uses the application framework to operate these applications [44].

- **Application Framework.** This provides many higher-level services to applications in the form of Java classes within an application. It also handled calls made by the application. Developers use this layer in building their applications and expand the components which are already present in Application Programming Interface.

- **Native Libraries and Android Runtime.** Above the Linux kernel is a set of libraries that transfer instructions to guide the device in handling different types of data. It consists of open-source Web browser engine Web Kit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data and SSL libraries responsible for Internet
REFERENCES


