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 termaju 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 sukar untuk mengenal pasti kecacatan-kecacatan pada keadaan yang berubah-ubah mengikut kecederungan 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 statik 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 ekperimen 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 mengesan kesalahan, ujian mutasi dilakukan dengan memperkenalkan mutan kepada aplikasi. Hasil daripada analisis mutasi menunjukkan bahawa 100% mutan terbunuh. Ini menunjukkan bahawa TEGDroid mempunyai keupayaan untuk mengesan kesalahan dalam aplikasi mudah alih.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
APK	-	Application Package File
ART	-	Android Run Time
AOR	-	Arithmetic Operator Replacement
C	-	Coverage
DVM	-	Dalvik Virtual Machine
EDS	-	Event Driven Software
Eh	-	Event Handler
GPS	-	Global Positioning system
GUI	-	Graphical User Interface
GW	-	GUI Widget
HLL	-	High Level Language
ICR	-	Incline Constant Replacement
ID	-	Event id
LCR	-	Logical Connector Replacement
LOC	-	Line of Code
MK	-	Mutant Killed
MS	-	Mutation Score
NOI	-	Negative Operator Inversion
OS	-	Operating System
ROR	-	Relational Operator Replacement
RVR	-	Return Value Replacement
SDK	-	Software Development Kit
SUT	-	Software Under Test

TM	-	Total Mutant
USB	-	Universal Serial Bus
W	-	Windows
WTG	-	Windows Transition Language
Wi-Fi	-	Wireless Fidelity
XML	-	Extensible Mark-up Language
TEGDroid	-	Test case generation approach for android apps

LIST OF APPENDICES

LIST OF PUBLICATIONS

Journal:

- Usman A., Ibrahim N. and Salihu I. A., TEGDroid: “A Test Case Generation Approach for Mobile Applications Considering Context and GUI Events”. International Journal of Pure and Applied Mathematics (IJPAM), SCOPUS indexed (Accepted).
- Usman, A., Ibrahim, N. and Salihu, I. A. (2018). “Comparative Study of Mobile Applications Testing Techniques for Context Events”. Advanced Science Letters, 24(10), 7305-7310.

Proceeding:

- Usman A., Ibrahim N. and Salihu I. A., “Test Case Generation from Android Mobile Applications Focusing on Context Events”. In Proceedings of the 2018 7th International Conference on Software and Computer Applications (pp. 25-30). ACM.

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 Andridmanifest.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.

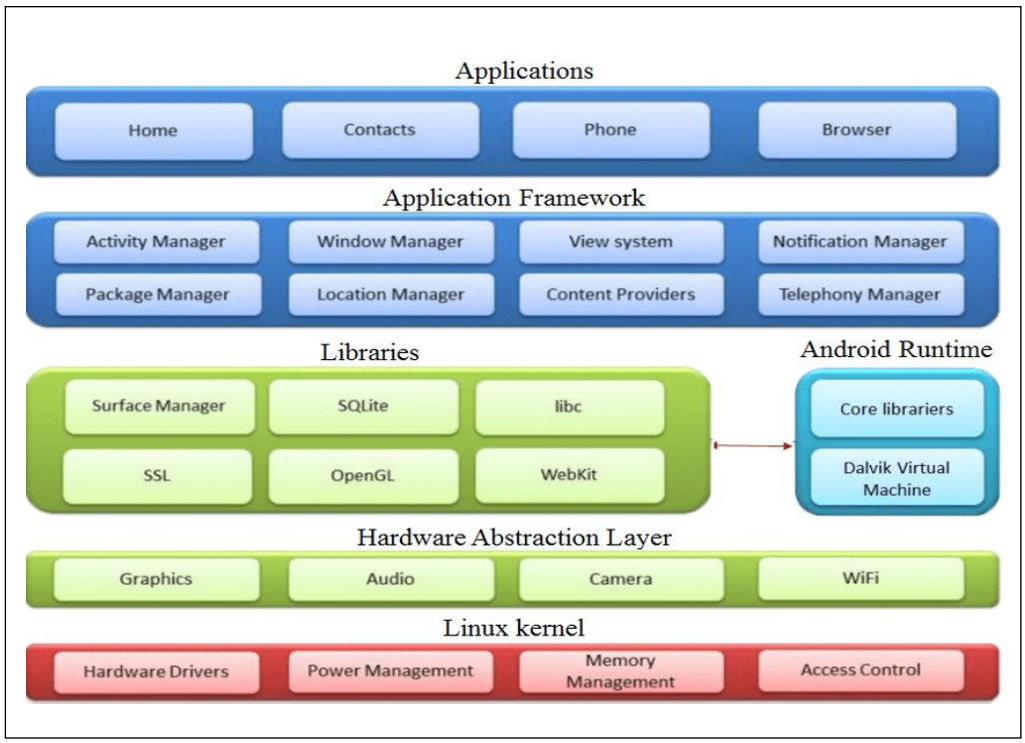

Figure 2.1: Android Framework Architecture [43]

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

1. Bala, K., Sharma, S. and Kaur, G. A study on smartphone based operating system. *International Journal of Computer Applications*. 2015. 121(1):17-22
2. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M. and Steggles, P. Towards a better understanding of context and context-awareness. *International Symposium on Handheld and Ubiquitous Computing*. September 27-29 Karlsruhe,Germany: Springer. 1999. pp. 304-307.
3. Nguyen, B.N., Robbins, B., Banerjee, I. and Memon, A. GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engineering*. 2014. 21(1): 65-105.
4. Liu, Z., Gao, X. and Long, X. Adaptive random testing of mobile application. *2nd International Conference on Computer Engineering and Technology (ICCET)*. 16-19 April. Chengdu, China: IEEE. 2010. pp. 297-301.
5. Amalfitano, D., Fasolino, A.R. and Tramontana, P. A gui crawling-based technique for android mobile application testing. *Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 21-25 March. Berlin: IEEE. 2011. pp. 252-261.
6. Amalfitano, D., Riccio, V., Paiva, A.C. and Fasolino, A.R. Why does the orientation change mess up my Android application? From GUI failures to code faults. *Software Testing, Verification and Reliability*. 2018. 28(1):1-27.
7. Tamilarasi, T. and Prasanna, M. Research and Development on Software Testing Techniques and Tools. In: Mehdi Khosrow-Pour, D.B.A. *Encyclopedia of Information Science and Technology*. 4th Ed. USA: IGI Global. pp. 7503-7513; 2018.
8. Myers, G.J., Sandler, C. and Badgett, T. *The art of software testing*. 2nd Ed. Hoboken, New Jersey: John Wiley & Sons. 2011.

9. Memon, A.M., Pollack, M.E. and Soffa, M.L. Using a goal-driven approach to generate test cases for GUIs. *Proceedings of the 1999 International Conference on Software Engineering*. 22-22 May. Los Angeles, USA: IEEE. 1999. pp. 257-266.

10. Iacob, I.M. and Constantinescu, R. Testing: First step towards software quality. *Journal of Applied Quantitative Methods*. 2008. 3(3): 241-253.

11. Berner, S., Weber, R. and Keller, R.K. Observations and lessons learned from automated testing. *Proceedings of the 27th international conference on Software engineering*. May 15 - 21. New York, USA: ACM. 2005. pp. 571-579.

12. Amaricai, S. and Constantinescu, R. Designing a Software Test Automation Framework. *Informatica Economica*. 2014. 18(1): 152.

13. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J. and McMinn, P. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*. 2013. 86(8): 1978-2001.

14. Yang, W., Prasad, M.R. and Xie, T. A grey-box approach for automated GUI-model generation of mobile applications. *International Conference on Fundamental Approaches to Software Engineering*. 16-24 March. Rome, Italy: Springer. 2013. pp. 250-265.

15. Fazzini, M., Freitas, E.N.d.A., Choudhary, S.R. and Orso, A. From Manual Android Tests to Automated and Platform Independent Test Scripts. *arXiv preprint arXiv:1608.03624*. 2016.

16. Aho, P., Suarez, M., Memon, A. and Kanstrén, T. Making GUI Testing Practical: Bridging the Gaps. *12th International Conference on Information Technology-New Generations (ITNG)*. 13-15 April. Las Vegas, Nevada, USA: IEEE. 2015. pp. 439-444.

17. Amalfitano, D., Amatucci, N., Tramontana, P., Fasolino, A.R. and Memon, A.M. A General Framework for comparing Automatic Testing Techniques of Android Mobile Apps. *Journal of Systems and Software*. 2016. 125: 322-343.

18. Amalfitano, D., Fasolino, A.R., Tramontana, P., De Carmine, S. and Memon, A.M. Using GUI ripping for automated testing of Android applications. *Proceedings of the 27th IEEE/ACM International Conference on Automated*

Software Engineering. September 03 - 07. Essen, Germany: ACM. 2012. pp. 258-261.

19. Azim, T. and Neamtiu, I. Targeted and depth-first exploration for systematic testing of android apps. *ACM SIGPLAN Notices*. ACM. 2013. pp. 641-660.
20. Hao, S., Liu, B., Nath, S., Halfond, W.G. and Govindan, R. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. June 16 - 19. Bretton Woods, NH, USA: ACM. 2014. pp. 204-217.
21. Takala, T., Katara, M. and Harty, J. Experiences of system-level model-based GUI testing of an Android application. *Fourth IEEE International Conference on Software Testing, Verification and Validation*. 21-25 March. Berlin: IEEE. 2011. pp. 377-386.
22. Amalfitano, D., Fasolino, A.R., Tramontana, P. and Amatucci, N. Considering context events in event-based testing of mobile applications. *Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 18–20 March. Luxembourg: IEEE. 2013. pp. 126-133.
23. Salihu, I.A., Ibrahim, R. and Mustapha, A..A Hybrid Approach for Reverse Engineering GUI Model from Android Apps for Automated Testing. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*. 2017. vol. 9 (3-3): 45-49.
24. Muccini, H., Di Francesco, A. and Esposito, P. Software testing of mobile applications: Challenges and future research directions. *7th International Workshop on Automation of Software Test (AST)*. June 02 - 09. Zurich, Switzerland: IEEE. 2012. pp. 29-35.
25. Morgado, I.C., Paiva, A.C. and Faria, J.P. Automated pattern-based testing of mobile applications. *9th International Conference on the Quality of Information and Communications Technology (QUATIC)*. September 23-26. Guimaraes, Portugal: IEEE. 2014. pp. 294-299.
26. Salihu, I.A. and Ibrahim, R. Comparative Analysis of GUI Reverse Engineering Techniques. *Advanced Computer and Communication Engineering Technology*. Springer. 2016. pp. 295-305.

27. Méndez-Porras, A., Quesada-López, C. and Jenkins, M. Automated testing of mobile applications: a systematic map and review. *XVIII Ibero-American Conference on Software Engineering*. Lima-Peru: CIbSE. 2015. pp. 195-208.

28. Griebe, T. and Gruhn, V. A model-based approach to test automation for context-aware mobile applications. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. March 24 - 28. Gyeongju, Korea: ACM. 2014. pp. 420-427.

29. Salihu, I.A. and Ibrahim, R. Systematic Exploration of Android Apps' Events for Automated Testing. *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*. November 28-30. Singapore: ACM. 2016. pp. 50-54.

30. Mirzaei, N., Garcia, J., Bagheri, H., Sadeghi, A. and Malek, S. Reducing combinatorics in GUI testing of android applications. *Proceedings of the 38th International Conference on Software Engineering*. May 14 - 22. Austin, TX, USA: ACM. 2016. pp. 559-570.

31. Amalfitano, D., Fasolino, A.R., Tramontana, P., Ta, B.D. and Memon, A.M. MobiGUITAR: Automated model-based testing of mobile apps. *IEEE Software*. 2015. vol. 32 (5): 53-59.

32. Hu, C. and Neamtiu, I. Automating GUI testing for Android applications. *Proceedings of the 6th International Workshop on Automation of Software Test*. May 21-28. Honolulu, HI, USA: ACM. 2011. pp. 77-83.

33. Su, T., Meng, G., Chen, Y., Wu, K., Yang, W., Yao, Y., Pu, G., Liu, Y. and Su, Z. Guided, stochastic model-based gui testing of android apps. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. September 04 - 08. Paderborn, Germany: ACM. 2017. pp. 245-256.

34. Song, K., Han, A.-R., Jeong, S. and Cha, S.D. Generating various contexts from permissions for testing Android applications. *27th International Conference on Software Engineering and Knowledge Engineering.* July 6-8. Pittsburgh, USA: *SEKE*. 2015. pp. 87-92.

35. Adamsen, C.Q., Mezzetti, G. and Møller, A. Systematic execution of android test suites in adverse conditions. *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. July 12-17. Baltimore, MD, USA: ACM. 2015. pp. 83-93.

36. Song, W., Qian, X. and Huang, J. Ehbdroid: beyond GUI testing for android applications. *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. October 30 to November 3. Illinois, USA: IEEE. 2017. pp. 27-37.

37. Machiry, A., Tahiliani, R. and Naik, M. Dynodroid: An input generation system for android apps. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. August 18-26. Saint Petersburg, Russia: ACM. 2013. pp. 224-234.

38. GitHub (2017). *GitHub*. Retrieved on Febuary 20, 2017, from https://github.com/

39. SourceForge (2017). *SourceForge*. Retrieved on March 09, 2017, from https://sourceforge.net/

40. Wasserman, A.I. Software engineering issues for mobile application development. *Proceedings of the FSE/SDP workshop on Future of software engineering research*. November 7-11. Santa Fe, NM, USA: ACM. 2010. pp. 397-400.

41. Businessweek (2017). *Bloombarg*. Retrieved on Febuary 14, 2017, from https://web.archive.org/web/20110205190729/http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.html.

42. Android developer (2017). *Intent*. Retrieved on September 30, 2017, from https://developer.android.com/guide/components/intents-filters.html.

43. Developer Android (2017). *Android platform*. Retrieved on September 02, 2017, from https://developer.android.com/guide/platform/index.html.

44. Kirandeep, A.G. Implementing security on android application. *The International Journal Of Engineering And Science (IJES)*. 2013. vol. 2 (3): 2319-1813.

45. Statista (2017). *Numbber of Availables Apps in Google Play Store*. Retrieved on Febuary 02, 2018, from https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

46. Statista (2017). *Mobile App Revenue*. Retrieved on October 17, 2017, from https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast.

47. Banerjee, I.. *Event-Code Interaction Directed Test Cases*. Ph.D Dissertation. University of Maryland; College Park. 2016.

48. Android developer (2016). *Apps Component*. Retrieved on December 06, 2016, from https://developer.android.com/guide/components.

49. Parada, A.G. and de Brisolara, L.B. A model driven approach for Android applications development. *2012 Brazilian Symposium on Computing System Engineering (SBESC)*. November 5-9. Natal, Brazil: IEEE. 2012. pp. 192-197.

50. SdxDevelopers (2016). *Application Package File (APK),*. Retrieved on November 30, 2016, from https://archive.is/20120717114627/ http:// forum .sdx-developers.com/index.php?topic=3472.0.

51. Justamomentgoose (2016). *Structure of APK*. Retrieved on November 30, 2016, from https://justamomentgoose.files.wordpress.com/2013/06/apk-structure.png.

52. Baldauf, M., Dustdar, S. and Rosenberg, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*. 2007. vol. 2 (4): 263-277.

53. Chen, G. and Kotz, D. A survey of context-aware mobile computing research. ed: Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College. 30 November 2000.

54. Chan, P.P.K. and Wen-Kai, S. Static detection of Android malware by using permissions and API calls. *International Conference on Machine Learning and Cybernetics*. July 13-16. Lanzhou, China: IEEE. 2014. pp. 82-87.

55. Wei, X., Gomez, L., Neamtiu, I. and Faloutsos, M. Permission evolution in the android ecosystem. *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012. pp. 31-40.

56. Developer Android (2016). *Android permission*. Retrieved on December 28, 2016, from https://developer.android.com/guide/topics/manifest/manifest-intro.html

57. Johnson, R., Wang, Z., Gagnon, C. and Stavrou, A. Analysis of android applications' permissions. *Sixth International Conference on Software Security and Reliability Companion (SERE-C).* June 20-22. Gaithersburg, Maryland: IEEE. 2012. pp. 45-46.

58. Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X.S. and Zang, B. Vetting undesirable behaviors in android apps with permission use analysis. *Proceedings of the 2013 ACM SIGSAC conference on Computer &*

communications security. November 04-08. Berlin, Germany: ACM. 2013. pp. 611-622.

59. Banerjee, I., Nguyen, B., Garousi, V. and Memon, A. Graphical user interface (GUI) testing: Systematic mapping and repository. *Information and Software Technology*. 2013. 55(10): 1679-1694.
60. Shahamiri, S.R., Kadir, W.M.N.W., Ibrahim, S. and Hashim, S.Z.M. An automated framework for software test oracle. *Information and Software Technology*. 2011. 53(7): 774-788.
61. Zhu, H., Hall, P.A. and May, J.H. Software unit test coverage and adequacy. *Acm computing surveys (csur)*. 1997. 29(4): 366-427.
62. Bertolino, A. Software testing research: Achievements, challenges, dreams. *Future of Software Engineering*. May 23-25. Minneapolis, MN, USA: IEEE. 2007. pp. 85-103.
63. Kaur, K. Survey of Software Test Case Generation Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2013. vol. 3 (6) :937-942.
64. ITpreneurpune (2016). *Manual Testing*. Retrieved on December 05, 2016, from http://itpreneurpune.com/challenges-with-manual-testing/.
65. Huizinga, D. and Kolawa, A. *Automated defect prevention: best practices in software management*. United States: John Wiley & Sons. 2007.
66. Grilo, A.M., Paiva, A.C. and Faria, J.P. Reverse engineering of GUI models for testing. *5th Iberian Conference on Information Systems and Technologies (CISTI)*. June 16-19. Santiago de Compostela, Spain: IEEE. 2010. pp. 1-6.
67. Campos, J.C., Saraiva, J., Silva, C. and Silva, J.C. GUIsurfer: A reverse engineering framework for user interface software. In: Alexandru, C. T. *Reverse Engineering-Recent Advances and Applications*. Netherlands: InTechOpen. 2012. pp. 31-54.
68. Kull, A. Automatic GUI model generation: State of the art. *23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)*. November 27-30. Dallas, Texas, USA: IEEE. 2012. pp. 207-212.
69. Moore, M.M. Rule-based detection for reverse engineering user interfaces. *Proceedings of the Third Working Conference on Reverse Engineering*. October 23-25. Benevento, Campania, Italy: IEEE. 1996. pp. 42-48.

70. Křoustek, J. and Kolář, D. Approaching Retargetable Static, Dynamic, and Hybrid Executable-Code Analysis. *Acta Informatica Pragensia*. 2013. vol. 2 (1): 18-29.

71. Mirzaei, N., Bagheri, H., Mahmood, R. and Malek, S. Sig-droid: Automated system input generation for android applications. *26th International Symposium on Software Reliability Engineering (ISSRE)*. November 2-5. Gaithersbury, Maryland, USA: IEEE. 2015. pp. 461-471.

72. Choudhary, S.R., Gorla, A. and Orso, A. Automated Test Input Generation for Android: Are We There Yet?(E). *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. November 09-13. Washington, DC, USA: IEEE. 2015. pp. 429-440.

73. Nguyen, C.D., Marchetto, A. and Tonella, P. Combining model-based and combinatorial testing for effective test case generation. *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. July 15-20. Minneapolis, MN, USA: ACM. 2012. pp. 100-110.

74. Ruiz, A. and Price, Y.W. Test-driven gui development with testng and abbot. *Ieee Software*. 2007. 24(3): 51-57.

75. Chen, T.Y., Kuo, F.-C., Merkel, R.G. and Tse, T. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*. 2010. 83(1): 60-66.

76. Majchrzak, T.A. and Schulte, M. Context-dependent testing of applications for mobile devices. *Open Journal of Web Technologies (OJWT)*. 2015. 2(1): 27-39.

77. Yu, S. and Takada, S. Mobile application test case generation focusing on external events. *Proceedings of the 1st International Workshop on Mobile Development*. March 10-12. Barcelona, Spain: ACM. 2016. pp. 41-42.

78. Yu, S. and Takada, S. External event-based test cases for mobile application. *Proceedings of the Eighth International Conference on Computer Science & Software Engineering*. July 13-15. Yokohama, Japan: ACM. 2015. pp. 148-149.

79. Liang, C.-J.M., Lane, N.D., Brouwers, N., Zhang, L., Karlsson, B.F., Liu, H., Liu, Y., Tang, J., Shan, X. and Chandra, R. Caiipa: Automated large-scale mobile app testing through contextual fuzzing. *Proceedings of the 20th annual*

international conference on Mobile computing and networking. September 7-11. Maui, HI, USA: ACM. 2014. pp. 519-530.

80. Mao, K., Harman, M. and Jia, Y. Sapienz: multi-objective automated testing for Android applications. *Proceedings of the 25th International Symposium on Software Testing and Analysis*. July 18-22. Saarbrücken, Germany: ACM. 2016. pp. 94-105.
81. Vieira, V., Holl, K. and Hassel, M. A context simulator as testing support for mobile apps. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. April 13-17. Salamanca, Spain: ACM. 2015. pp. 535-541.
82. Inozemtseva, L. and Holmes, R. Coverage is not strongly correlated with test suite effectiveness. *Proceedings of the 36th International Conference on Software Engineering*. May 31- June 07. Hyderabad, India: ACM. 2014. pp. 435-445.
83. Deng, L., Offutt, J., Ammann, P. and Mirzaei, N. Mutation operators for testing Android apps. *Information and Software Technology*. 2017. 81: 154-168.
84. Ammann, P. and Offutt, J. *Introduction to software testing*. New york: Cambridge University Press. 2016.
85. Delgado-Pérez, P., Medina-Bulo, I. and Núñez, M. Using Evolutionary Mutation Testing to improve the quality of test suites. *Congress on Evolutionary Computation (CEC)*. June 5-8. Donostia - San Sebastián, Spain: IEEE. 2017. pp. 596-603.
86. DeMillo, R.A., Lipton, R.J. and Sayward, F.G. Hints on test data selection: Help for the practicing programmer. *Computer*. 1978. 11(4): 34-41.
87. Deng, L., Offutt, J. and Samudio, D. Is Mutation Analysis Effective at Testing Android Apps? *International Conference on Software Quality, Reliability and Security (QRS)*. July 25-29. Prague: IEEE. 2017. pp. 86-93.
88. Moran, K., Tufano, M., Bernal-Cárdenas, C., Linares-Vásquez, M., Bavota, G., Vendome, C., Di Penta, M. and Poshyvanyk, D. MDroid+: a mutation testing framework for android. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. May 27 - June 03. Gothenburg, Sweden: ACM. 2018. pp. 33-36.
89. Github (2017). *muJava*. Retrieved on August 16, 2017, from https://github.com/jeffoffutt/muJava.

90. Wei, Y. *MuDroid: Mutation Testing for Android Apps*. Final Year Project. University College London. 2016.

91. Research Appendix (2017). *mdroid+* Retrieved on October 15, 2017, from https://research-appendix.com/mdroid/.

92. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. *Experimentation in software engineering*. New York: Springer Science & Business Media. 2012.

93. Android developer (2017). *Monkeyrunner*. Retrieved on May 29, 2017, from http://developer.android.com/tools/index.html.

94. Bartel, A., Klein, J., Le Traon, Y. and Monperrus, M. Dexpler: converting android dalvik bytecode to jimple for static analysis with soot. *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis*. June 11 -16. Beijing, China: ACM. 2012. pp. 27-38.

95. Github (2017). *Apktool*. Retrieved on January 15, 2017, from https://ibotpeaches.github.io/Apktool/.

96. Wichmann, B., Canning, A., Clutterbuck, D., Winsborrow, L., Ward, N. and Marsh, D. Industrial perspective on static analysis. *Software Engineering Journal*. 1995. 10(2): 69-75.

97. Presto (2017). *GATOR: Program Analysis Toolkit For Android* Retrieved on November 9, 2016, from http://web.cse.ohio-state.edu/presto/software/gator/.

98. Adamo, D., Nurmuradov, D., Piparia, S. and Bryce, R. Combinatorial-based event sequence testing of Android applications. *Information and Software Technology*. 2018. 99: 98-117.

99. Li, X., Gao, R., Wong, W.E., Yang, C. and Li, D. Applying combinatorial testing in industrial settings. *International Conference on Software Quality, Reliability and Security (QRS)*. August 1-3. Vienna, Austria: IEEE. 2016. pp. 53-60.

100. Brader, L., Hilliker, H. and Wills, A.C. *Testing for Continuous Delivery with Visual Studio 2012*. 1st Ed. USA: Microsoft patterns & practices. 2012.

101. SEMaterial (2017). *Test Coverage with EclEmma*. Retrieved on August 19, 2017, from http://realsearchgroup.org/SEMaterials/tutorials/eclemma/.

102. SourceForge (2017). *Emma, An open source Java code coverage tool*. Retrieved on November 30, 2017, from http://emma.sourceforge.net/.

103. Gay, G., Staats, M., Whalen, M. and Heimdahl, M.P. The risks of coverage-directed test case generation. *IEEE Transactions on Software Engineering*. 2015. 41(8): 803-819.

104. Gopinath, R., Jensen, C. and Groce, A. Code coverage for suite evaluation by developers. *Proceedings of the 36th International Conference on Software Engineering*. May 31 - June 07. Hyderabad, India: ACM. 2014. pp. 72-82.

105. SourceForge (2017). *BarcodeScanner*. Retrieved on April 11, 2017, from https://beem-project.com/attachments/157/Beem-0.1.8.apk.

106. GitHub (2017). *Beem* Retrieved on April 11, 2017, from https://beem-project.com/attachments/157/Beem-0.1.8.apk.

107. Pierrox (2017). *MarineCompass* Retrieved on April 13, 2017, from http://www.pierrox.net/cmsms/applications/marine-compass.

108. SourceForge (2017). *OpenCamera* Retrieved on March 30, 2017, from https://sourceforge.net/projects/opencamera/files/v_1_42/.

109. GitHub (2017). *Pedometer* Retrieved on April 10, 2017, from https://github.com/bagilevi/android-pedometer.

110. SourceForge (2017). *Subsonic* Retrieved on April 10, 2017, from https://sourceforge.net/projects/subsonic/files/android/4.4/.

111. GitHub (2017). *TippyTipper* Retrieved on April 14, 2017, from https://github.com/mandlar/tippytipper/tree/master/.

112. Googleplay (2017). *Beem Install* Retrieved on December 19, 2017, from https://play.google.com/store/apps/details?id=com.beem. project.beem.