The Framework for Android Apps Testing using Mutation Technique

Muhammad Arshad Javed\textsuperscript{1a}, Rosziati Ibrahim\textsuperscript{1b}, Jahari Abdul Wahab\textsuperscript{2c}

\textsuperscript{1}Faculty of Computer Science and Information Technology, UniversitiTun Hussein Onn Malaysia, Parit Raja, 86400 Johor, Malaysia. 
\textsuperscript{2}Department of Engineering R&D, SENA Traffic Systems Sdn. Bhd., Kuala Lumpur, Malaysia.

\textsuperscript{a}arshadbwp@gmail.com; \textsuperscript{b}rosziati@uthm.edu.my; \textsuperscript{c}jahari@senatraffic.com.my

Abstract. This paper presents the framework for Android apps testing using mutation technique. A systematic comparative analysis is done based on various approaches and techniques for formulation testing. The analysis examines the development of research on this subject and its inspiring objectives and practices. Using comparative analysis, the framework exhibits that Android mutation testing can create assessment cases that are very efficient at distinguishing both naturally happening errors and crowd-sourced errors for Android apps. The framework can be used as a basis to test the Android apps using mutation technique.

1. Introduction

Software programs which are installed and used on different electronic gadgets such as tablets, smart televisions, smart phones and smart watches are called smart applications (apps). As the prices of these smart electronic devices are regularly progressing low and the mobile network systems are swiftly growing in different parts of the world, the number of users of these devices have been increasing. Around the world more people are getting these smart devices. Sometime in the daily life, a single person may keep more than one mobile gadgets. He may carry a smart cellular phone to communicate with his friends and family members. He may have smart television at home or at office for watching news programs, movies and sports and uses smart watch for following movements and other activities.

Recently, Cisco reported that in 2018 the number of cellular devices reached 8.0 billion in all over the world. This number of devices is more than the entire population (7.4 billion) of this world at the same time [1]. It is not surprised, mobile gadgets/appliances have been used more commonly than desktop computers and laptops by the users who belong to different fields of life. Since 2008, a USA based venture investment company, has followed the time that Americans spend on their electronic devices, cellular phones, iPads, laptops and desktops. Report presented by Mary Meeker [2] in 2018 stated that on average, Americans spend 2.8 hours per day on their electronic devices as compared to only 0.4 hours per day in 2010. Mobile gadgets have developed from sluggish devices with a low, less-resolution, black and white display screens which are efficient of just voice phoning, text messaging and looking news to prevailing portable computers and entertaining visual display units. Lifestyle of the people have been changed by mobile devices, for example online shopping (e-commerce) where people purchase and sell their products, entertainment, travel and communication with friends and make new friends through social media (e-mail, twitter, Instagram, Facebook). These things became very critical to the people’s everyday life. A study accomplished by TechRepublic in 2016 showed that 94% of the respondents used Mobile gadgets at their workplace [3].

Android apps have distinctive characteristics and additional attributes that differ from the traditional software. Most of the Android apps are created in Java. The advanced features and characteristics
create the difficulty in testing the Android apps. This brings the challenges to the programmers and IT professionals when testing the Android apps.

2. Related Work
Android operating system is different than other types of software. It needs major elements of these apps to perform according to a pre-described life span [4]. If an elements lifecycle is not properly managed, it is more probably to trigger unexpected problems when the users switch between different applications, stop then restart an app or activate the device from sleep mode. Especially the stream of continuity in most of the Android apps is very important to the users like games. Testing mobile apps will also be different from web applications [5].

As compare to activity that presents a screen to the users, services are unseen and accomplish long term execution tasks, like background music. The services also act with respect to the predefined lifecycle and it is different than activity’s lifespan. Subsequently the services and other elements must communicate with each other to execute Android apps smoothly, especially for those apps which need the accurate executing of services, for example melody players, alarm clocks and e-mail clients, programmers must properly implement the services according to their lifecycle. Study of alarm-clock app found that software bugs in its services that triggered the app to sound at incorrect time lead to runtime fails.

Although most of the source code of Android apps is written in Java programming language, for program configuration, XML files are also used intensively in Android apps, layout and provisional data storage. According to the design and requirements of the projects, Android app can include more XML files than Java source code files. Such as the programs with graphical user interface (GUI) applied using Java or C# hardly use XML files. Therefore, no testing methods designed for testing conventional Java programs consider the source code other than Java in the same task. Moreover, there is no assessment handling standard to measure this assessment information for XML data/files employed in Android apps. Certainly, it may produce unexpected catastrophes without evaluating XML files of Android apps. Rules for XML schema have been discussed in [6].

In a study done by Lisper et al. [7], they emphasized that the embedded systems became very important part of the daily life. In the environment they execute very important functions and they are required more than desktop systems especially the non-functional characters, for example operation, proficiency, robustness, security and trustworthiness are becoming progressively very important. So, it is required to have excellent techniques for validating these characteristics. The important assumption motivating their method is control flow (iterations), which strongly influence the changeability of the accomplishment time of the program. They confine mutations to perform simply in such portions of the program/code that may disturb the stream of the entire program.

Mutation testing is generally considered as more test criterion because it creates a massive number of mutants. Even though several attempts have been endeavoured to decrease the cost of computation, but the scalability matter remains there in practice [8]. An important advantage of mutation testing stated in the literature is that it presents an improvement in the test of the error finding ability of test sets as evaluated to some other test reporting standards. Regardless of its renowned advantages, mutation testing continuously is a very costly activity subsequently it needs to accomplish the assessment sets against every mutant, whose number promote exponentially along through the volume of code under the assessment. Zhu et al. [9] focused their investigation around six strategies of mutation compression which they planned. More precisely, they adopted coincided classifying and Formal Concept Analysis (FCA) to group mutants and assessment sets built on the program reporting and required (ineffective mutation) circumstances. Weak mutation applies the required condition; a mutant is destroyed if its accomplishment directs to a state alter. FCA was an approach of data
assessment and it is proved to be a strong arithmetical method to carry out and summarize an enormous volume of data. In this study the researchers followed a six steps compression strategy. This compression strategy is shown in Table 1 [9].

Table 1: Mutation Data Compression [9]

| No | Instrumentation                              | Test Execution                                                                 | Reachability and requisite analysis | Mutant Clustering                          | Data Compression                                          | Mutant Execution                                      |
|----|----------------------------------------------|--------------------------------------------------------------------------------|-------------------------------------|--------------------------------------------|----------------------------------------------------------|--------------------------------------------------------|
| 1  | The actual code/program is instrumented to retain track of mutation location. Technique of mutant schema is applied. | The required suite of test is executed on the actual program. They recorded the mutants which are touched and weakly killed by the tests. | The output/results are saved and analyzed in two matrices known as mutant by assess reachability and mutant by test necessary respectively. | They applied clustering technique to cluster similar mutants together and used two techniques of clustering, overlapped grouping and FCA grouping. | They compressed the mutant clusters. Mutation knowledge is used during the compression process to achieve accurate result. | A strong analysis is carried out on compressed matrix. |

Model based methods have been expanded to assess Android apps also, like using state machines and graphical user interface. But the availability of the models is one pre requirement of model-based methods. It is very difficult to perform model-based testing in the absence or unsatisfactory availability of the models. Moreover, it is very common that abstracts of different models can be produced by different people for the similar app [9].

Some researchers expand symbolic execution into Android apps testing. Symbolic execution of Android apps is not easy as it contains either developing an adapted virtual machine for Android devices or exhibiting the Android libraries [10]. Other researchers use refactoring algorithm for testing the android apps [11-12]. Test cases can also be generated from system’s requirements [13]. There is another non-functional problem for Android devices, called energy inadequacy causing proliferated concerns from the users. Some available methods notice energy inefficiency issues by discovering application states with Java path finder (JPF) structure. But the results of these methods are contained because of their indefinite application performance models and inadequate energy inefficiency designs. Navy Droid is an effective tool introduced by Liu to diagnose energy inefficiency issue for Android apps. At present, few of the suggested methods are dynamically maintained or employed by industry in real Android apps [14].

Random assessing of Android apps is appealing because of its easy use and scalability, but its usefulness could be distrusted. Monkey, a simple technique and tool for random assessing of Android apps, is amazingly effective and much more refined tools by getting more attention [15]. Table 2 summarizes the related works that have been used as a basis to form the framework in testing the Android apps.

Table 2: Summary of Related Work

| No | Authors | Heuristic Technique | Details |
|----|---------|---------------------|---------|
| 1  | Loiseet al. 2017 [14] | Security aware mutation operators and PIT mutation engine | With the use of their mutants, security concerned test suites can be created and documented. Their mutation operators can be used in education and in systematic evaluation and comparison of fuzzing |
and other security related testing tools.

|   | Authors                          | Methodology                                                                 | Results/Outcome                                                                 |
|---|----------------------------------|-----------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| 2 | Iyama et al. 2018 [15]           | Inevitably creating test scripts by using static and dynamic analysis       | As the outcome of an evaluation test, it is shown that the presented approach can reduce the number of man-hours by about 61% compared with the traditional method of producing test scripts manually. |
| 3 | Lisper et al. 2017 [7]           | Targeted mutation testing of non-functional properties, program slicing     | They examine the grouping of static and mutation analysis for testing non-functional properties. They focus on worst-case execution time (WCET). |
| 4 | Zhu et al. 2017 [8]              | They followed Formal Concept Analysis (FCA) to collect related mutants together | The mutation testing process made use of data compression methods based on state infection information, promising solution to decrease the cost of mutation testing. |
| 5 | Zhu et al. 2018 [9]              | Mutant compression techniques                                               | The research showed that mutant compression approach is a better option than random sampling and weak mutation in exercise: they can efficiently speed up strong mutation 6.3 to 94.3 times with correctness of more than 90%. |
| 6 | Ibrahim et al. 2020 [12]         | Code smell and refactoring                                                 | The research uses code smell and refactoring techniques to reduce the redundancy of generating the test cases. |
| 7 | Bashir et al. 2017 [16]          | Improved genetic algorithm (two ways crossover and adaptable mutation approaches in eMuJava 2.0) | In this research, Experiments showed that improved algorithm attained the required objectives in lesser iterations and raised mutation score by identifying errors in the program. |
| 8 | Kim et al. 2018 [17]             | Invasive Software Testing (IST) technique, DEMINER                         | The research presented that DEMINER can effectively improve test handling by using DEMINER to three actual world programs in C. |
| 9 | Petrovic et al. 2018 [18]        | Mutation testing analysis at industrial level                             | This paper describes lessons from an industry progression use of mutation testing, classifies challenges that require to be directed, and proposes research guidelines to advance the implementation of mutation testing in exercise. |
| 10| Salihu et al. (2019) [19]        | Used static and dynamic approach for mobile apps                           | AMOGA is used to test the mobile applications using static and dynamic approach for the GUI testing. |

### 3. The Framework

Mutation investigation cannot be precisely directed to Android apps in similar way it is on conventional Java programs, since Android apps have diverse programming construction and are established, mounted and tested in diverse ways. Conventionally, Java mutation investigation tools either mutate and the source code of Java and accumulate to class files of bytecode, or at first accumulate to files of bytecode session, or foremost compile to bytecode then modify the bytecode.
Later that files of Java bytecode are vibrantly connect by the language system all through the running process. But there is an extra need in Android apps that every Android distorted should be compiled as a file of Android application package to facilitate, it can be installed and accomplished on mobile apparatuses and emulators. Similarly, since Android apps rigorously use XML files for program structure, blueprint design and description, this study purposes Android mutation operators to transform XML files too. These aspects expressively change the procedure, design and employment of Android mutation assessing tools. Figure 1 shows the process of mutation analysis for the Android apps.

![Mutation Analysis on Android Apps](image)

**Figure 1: Mutation Analysis on Android Apps**

Based from Figure 1, some of the steps are unique from conventional Java mutation testing procedures. The operator first chooses which mutation operators to be employed. In this research, we
will use different Android mutation analysis tools, known as muDroid which contains new Android mutation operators, some conventional Java mutation operators from MuJava and some removal operators. Analysis tool of Android mutation expands part of MuJava mutant creation engine to apply all these mutation operators.

For conventional Java mutation operators, the analysis tool transforms the actual Java files corresponding to stated rules of mutation and lists them to bytecode. The mutation operators related to XML are directed to XML files, producing new versions of each files as per XML mutants. Later Android mutation testing tool swaps this file into place to make it ready for dynamic tying when developing APK files.

The mutation procedure chooses a mutated Java bytecode-class file, integrates with other propose files and produces a mutated APK file as a distorted of the Android app in the test. Some mutants may trigger the compilation bugs. Additionally, there are numerous outward Android challenging automation frameworks like Robotium, Espresso and Selendroid, are commonly used by investigators/researchers to present automated assessing for Android apps. Android mutation testing adjusts all these assessing frameworks. Assessment cases can be prepared by the researchers with Android challenging automation frameworks to destroy mutants. Subsequently producing mutants and listing them to APK files, the system inserts the actual app under the test into the device. All test cases are compiled and run by the system on the apps and note the outputs like projected results. After that APK mutant file is inserted into the Android device. Mutation system runs all challenging cases aligned with the mutants and records the outputs being actual results.

Gathering all the results on Android investigation tool will then compare the projected results with real results. If the real results of the test vary from projected results of the similar test, then that mutant is recorded as being destroyed by that test. At the end, Android mutation investigation tool calculate the mutation enough result, the proportion of mutants destroyed by the tests to the complete number of non-alike mutants. The instrument does not employ any heuristics to assist find out equivalent mutants, therefore investigator requires to remove equivalent mutants himself manually.

If pre-described error models are existing, each kind of mistake in the error models can be employed to describe a mutation operator that can produce occurrences of the error. There is substitute method to investigate each syntactic component of the programming language being altered and create mutants to adapt the composition in means that standard programmers may be do blunders. There is no pre-described error model available. This research will conduct a study to find out some common errors in Android app development. In this research we will design and create seventeen Android mutation operators for Android apps and fix the exclusive assessing challenges.

4. Conclusion and Future Works

Android mobile machines and Android apps govern the worldwide market with respect of its users, developers, machines, and applications (apps). However, this size makes the worth issues of Android apps much severe. Worse software breakdowns are commonly noticed in many Android apps, for example runtime break downs, inappropriate behaviours, and security susceptibilities. Therefore, we badly need more state-of-the-art and efficient testing. This study examined the programming background, distinctive programming characteristics, and innovative features of Android apps, and established Android mutation assessing, an additional sophisticated testing method than existing practice. Unnecessary or unproductive Android test cases can be cleaned out, and eventually, the capability to bring excellence Android apps can be enriched. In this research we designed and proposed an Android mutation testing approach. The next step is to conduct the experiment on the test suite in demonstrating that the approach is very efficient at creating extraordinary feature test cases and testing already existing test assignments. Our future works include implementing the approach and test the approach in its effective manners.
Acknowledgments
This project is funded by the Ministry of Education Malaysia under the Malaysian Technical University Network (MTUN) grant scheme Vote K234 and SENA Traffic Systems Sdn. Bhd.

References
[1] Cisco (2018). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.pdf, (access on November 2019).
[2] Mary Meeker (2018). Internet trends 2018. Online, May 2018. https://www.kleinerperkins.com/perspectives/internet-trends-report-2018/, (access on November 2019).
[3] TechRepublic (2016). Infographic: Byod is popular, but not widely supported. http://www.techrepublic.com/article/infographic-byod-is-popular-but-not-widely-supported-by-it/, (access on November 2019).
[4] Ross, A. S., Zhang, X., Fogarty, J., & Wobbrock, J. O. (2018). Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility (pp. 119-130). ACM.
[5] Ahmed, M., Ibrahim, R. (2015). A Comparative Study of Web Application Testing and Mobile Application Testing. In 2015 Lecture Notes in Electrical Engineering (LNEE). 315, pp. 491-500.
[6] Aman, H., Ibrahim, R. (2014). Formalization of Transformation Rules from XML Schema to UML Class Diagram. In International Journal of Software Engineering and its Applications, 8 (12), pp. 75-90.
[7] Lisper, B., Lindström, B., Potena, P., Saadatmand, M., & Bohlin, M. (2017). Targeted mutation: Efficient mutation analysis for testing non-functional properties. In 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 65-68). IEEE.
[8] Zhu, Q., Panichella, A., & Zaidman, A. (2017). Speeding-up mutation testing via data compression and state infection. In 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 103-109). IEEE.
[9] Zhu, Q., Panichella, A., & Zaidman, A. (2018). An investigation of compression techniques to speed up mutation testing. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) (pp. 274-284). IEEE.
[10] Cao, Y., Wu, G., Chen, W., & Wei, J. (2018). CrawlDroid: Effective Model-based GUI Testing of Android Apps. In Proceedings of the Tenth Asia-Pacific Symposium on Internetware (p. 19). ACM.
[11] Ahmed, M., Ibrahim, R., Ibrahim, N. (2015). An Adaptation Model for Android Application Testing with Refactoring. In International Journal of Software Engineering and Its Application, 9 (0), pp. 65-74.
[12] Ibrahim, R., Ahmed, M., Nayak, R, Jamel, S. (2020). Reducing redundancy of test cases generation using code smell detection and refactoring. In Journal of King Saud University – Computer and Information Sciences, 32 (2020), pp. 367-374.
[13] Ibrahim, R., Saringat, M.Z., Ibrahim, N., Ismail, N.: An Automatic Tool for Generating Test Cases from System’s Requirements. CIT 2007: 7th IEEE International Conference on Computer and Information Technology, Article number 4385193, Pages 861-866, 2007.
[14] Loise, T., Devroey, X., Perrouin, G., Papadakis, M., & Heymans, P. (2017). Towards security-aware mutation testing. In 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 97-102). IEEE.
[15] Iyama, M., Kirinuki, H., Tanno, H., & Kurabayashi, T. (2018). Automatically Generating Test Scripts for GUI Testing. In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 146-150). IEEE.

[16] Bashir, M. B., & Nadeem, A. (2017). Improved genetic algorithm to reduce mutation testing cost. IEEE Access, 5, 3657-3674.

[17] Kim, Y., Hong, S., Ko, B., Phan, D. L., & Kim, M. (2018). Invasive software testing: Mutating target programs to diversify test exploration for high test coverage. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) (pp. 239-249). IEEE.

[18] Petrovic, G., Ivankovic, M., Kurtz, B., Ammann, P., & Just, R. (2018, April). An industrial application of mutation testing: Lessons, challenges, and research directions. In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 47-53). IEEE.

[19] Salihu, I.A., Ibrahim, R., Ahmed, B.S., Zamli, K.Z. Usman, A. (2019). AMOGA: A Static-Dynamic Model Generation Strategy for Mobile Apps Testing. IEEE Access. 2019. DOI:10.1109/ACCESS.2019.2895504