Test Case Generation from Android Mobile Applications Focusing on Context Events

Asmau Usman
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia, (UTHM)
Batu Pahat, Johor, Malaysia
gi160007@siswa.uthm.edu.my

Noraini Ibrahim
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia, (UTHM)
Batu Pahat, Johor, Malaysia
noraini@uthm.edu.my

Ibrahim Anka Salihu
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia (UTHM)
Batu Pahat, Johor, Malaysia
hi130015@siswa.uthm.edu.my

ABSTRACT
Nowadays mobile apps are developed to address more critical areas of people's daily computing needs, which bring concern on the applications' quality. Today's Mobile apps processed not only the traditional GUI events but also accept and react to constantly varying context events which may have an impact on the application's behaviour. To build high quality and more reliable applications, there is a need for effective testing techniques to test apps before release. Most of recent testing technique focuses on GUI events only making it difficult to identify other defects in the changes that can be inclined by the context in which an application runs. This paper proposed an approach for testing mobile apps considering the two sets of events: GUI events which we identified through static analysis of bytecode and context events obtained from analysis of manifest.xml file. Results from the experimental evaluation indicated that our approach is effective in identifying and testing context events.

CCS Concepts
• Software and its engineering • Software and its engineering→ Software defect analysis • Software and its engineering→ Context specific languages

Keywords
Software Testing; Android; GUI Event; Mobile Application; Context Event; Android Permissions; Test Case Generation.

1. INTRODUCTION
Smartphones are becoming increasingly popular in recent years. They are now widely used by many people for several computational tasks due to their increase in functionality and compatibility [1]. Android has become the most popular operating system for several mobile devices including smartphones [2]. The increase in rich innovative features that support mobility and access to phone hardware such as sensors, camera, cellular networks and Wi-Fi networks in smartphones has enabled the development of mobile applications (mobile apps) that can constantly monitor and react to their environment, and provide rich context-aware content to users [3-5]. These set of app process inputs from both users and constantly changing contexts [6]. Mobile apps are now developed to address more critical areas of people's daily computing needs, which brought concern on the applications' quality. To build high quality and more reliable applications that can gain recognition in the high-level competitive application's (app) market, there is a need for effective testing techniques to validate the quality of the applications. This technique should be able to validate different events supported by mobile apps [7] in order to improve users' confidence on the mobile apps [8-10]. Testing event-driven applications presents 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 [11].

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 [11, 12]. These changes can expand the set of test cases for testing a mobile app significantly [13].

Testing mobile apps external (context) events have numerous challenges. The major challenges are how to identify the external events from an application and how to trigger them during testing [6, 14]. 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 [8, 9, 15-19] without sufficient support for testing context events [12]. Therefore, it is difficult to identify other defects in the changes of the contexts which an application runs [4]. In order to ensure that these applications behave correctly, external context events must be considered during testing such as GPS location data, sensors e.t.c., in addition to the GUI events.

There are few techniques that considered context events [4, 6, 20], but the precise event sequences are generated based on a restricted number of scenarios. With the complexity and diverse sources of external events supported by mobile apps, the techniques are still incapable of supporting all the different contexts (e.g., those from wide system events). For instance, some apps cannot functions properly in low battery, as such when the app is launch in low battery, the user need to be notified. Another scenario is the

ICSCA 2018, February 8–10, 2018, Kuantan, Malaysia

ACM ISBN 978-1-4503-5414-1/18/02…$15.00
https://doi.org/10.1145/3185089.3185099

reception of a phone call or SMS that can significantly change the state of an app. This study focuses on such cases.

The paper proposed an approach for test case generation from mobile apps considering both GUI and context events for effective testing by extending the approach in [4]. The paper is organized as follows. Section 2 presents related works. Section 3 discusses the overview of context events and mobile apps permission. Section 4 proposes an improved approach for generating context events from mobile apps. A discussion was presented in Section 5 and section 6 presented the conclusion and our future work.

2. RELATED WORKS

The proposed approach aims at generating test cases from mobile apps considering context events using app's permission and GUI events by analyzing the GUI and Intent messaging mechanism. We summarize literature related to mobile apps GUI testing and contexts events testing. The goal is to place our work in the perspective of either inspiring or very recent papers in each area.

2.1 GUI Testing

Several testing techniques for mobile apps have evolved over the years. AndroidRipper [15] uses a Ripper that automatically explores an application's user interface with the aim of exercising the application behaviour in a structured manner. It focuses on user events from the GUI. Liu et. al [21] proposed a framework for automatic testing of Android apps based on the Capture/Replay method. The user events are captured and converted into Robotium test scripts that can be executed to replay the recorded actions of users. Automated model generator for android (AMOGA) [22] is a model based testing approach targeting GUI events. The generated model represents the behaviour of a mobile app which is used to generate test cases for testing an app. TrimDroid [17] is an automated approach for generating GUI system tests for Android apps using combinatorial testing technique which relies on program analysis to extract formal specifications. The objective of the approach is to reduce the number of tests in GUI testing of Android apps without compromising on coverage. Morgado et. al. [11] uses reverse engineering and behavioural pattern to automatically generate test cases that identifies and tests behaviours that are common in mobile applications. These approaches do not explore context behaviour of mobile apps. Therefore, they cannot be used to test mobile apps that react to context events.

2.2 Testing Context Events

There are few techniques available in the literature that considers context events to automatically generate test case. Smart-monkey [10] is an approach for testing mobile apps that combine elements of event-based testing and random testing to automatically generate test case for mobile apps. It uses an extended FSCS-ART technique proposed by [23]. Test cases are composed by sequence of both user events and context events based on event sequence distance and ART that is used on other event-driven software. The aim of the approach is to reduce the number of test cases and the time needed to expose first fault.

Amalfitano et. al. [6] proposed Extended Ripper that considers both context events and GUI events for testing Android mobile apps. It is based on reusable event patterns that was manually define after an initial analysis conducted on the bug reports of open source applications. Test cases can be generated based on the defined event patterns using three scenario-based mobile testing approaches: manual, mutation based and exploration based. Since the event patterns are derived manually by expert from analyses of bug history, the events that may trigger a faulty behavior in an app may not be identified accurately. This is because, sequence of events that has never occurred previously might not be selected, and it could be a source of unpleasant failures. Moreover, when testing other types of applications, these event patterns may need to be redefined.

Majchrzak and Schulte proposed Android tool [24] that addresses the context changes in mobile apps called block-based context-sensitive testing. Test cases are split in blocks that can be reused and combined with context changes that are used in testing different scenarios without duplicating the test cases. This reduces the effort of writing test cases. However, the approach cannot deal with the process steps that have to be executed in certain context. Yu and takada [20, 25] proposed an approach for test cases generation that consider both GUI and context events for android mobile apps using events pattern that is manually derived, similar to the one in [6].

Song et. al. [4] presented an approach that systematically generates several executing context from permission of android mobile apps. The approach analyses the lists of permissions where the resources that an application uses can be identified easily. Various contexts of an application are generated by permuting resource conditions, and the permutations of the contexts are prioritized and selected for test case generation. Few selected apps' permissions were considered to identify their related resources and possible states. As such some faulty behaviour could not be detected and there is no clear means of detecting context from dynamically changing environment.

THOR tool [14] is a test execution framework that considers certain system events in order to detect faults. However, the tool does not generate test cases but it uses test cases that are constructed manually using script-based technique. Dynodroid tool [26] uses Adaptive Random technique (ART) to generate sequence of events that can be used to systematically explore an application. The approach is based on observe-select-execute sequence to generate both user and context events by checking the ones that are relevant for the app. One of the limitations of the approach is restriction of the apps under test from communicating with other apps. As many Android apps communicate with other apps for shared functionality, some context could not be detected.

3. OVERVIEW OF CONTEXT EVENTS AND APPS' PERMISSION

This section presents the background of mobile apps context events. It further discusses Android Intent message system and app's permission. We leverage this knowledge and concepts in our proposed approach in the next sections.

3.1 Mobile App's External Events

Mobile apps receive and process two types of event: user GUI events and events due to external input sources such as system events and environmental contexts. User GUI events such as keyboard events and touch events are triggered through user's interaction with the application. While for the environmental context events, the operating system generates these events in response to solicitations from external sources, such as device sensors, timers, batteries, USB drivers, network details, and other sources within the system.

Context events triggered by external sources from the application can be grouped as follows [4, 6, 10]:

- Events coming from the external environment and sensed by device sensors (like temperature, pressure, GPS, geomagnetic field sensor, etc.);
- Events generated by the device hardware platform (such as battery and other external peripheral port, like USB, headphone, network receiver/sender, etc.);
- Events typical of mobile phones (such as the reception of SMS message and phone calls);
- Events generated from social networks notifications (such as nearby MSN friends, the current activity the user is up to, and even the user's mood).

In Android, context (external) events are handled using the Intent messaging object. Intent is an abstract description of an operation to be performed and it is used to request an action from another app component [27]. It is used to facilitate communication between components e.g., starting an activity, starting a service and delivering broadcast. Two types of intents have been defined in Android: explicit and implicit intent. To know about system wide events, a broadcast receiver needs to be implemented and registered. The system delivers various broadcasts for system events by passing an intent message [27].

3.2 Android Permission System

The permission-based model in Android operating system (OS) is implemented by Google to control access to sensitive components and data in the android smartphone [1]. The OS uses the permission-based security mechanism to control the behaviour of applications such as access to sensitive data (e.g., contacts, call logs or photos) and to notify the user of potential dangerous device behaviours (e.g., GPS, camera and internet) [28]. Any application that needs access to certain resource or components protected by the permission must declare the corresponding permission in the application by listing the permissions in the manifest file [29]. During installing, a user will be informed of the permissions required by the application that are listed in the manifest file for confirmation. Numerous researches have discussed using Android permissions for malware detection such as [28, 30-32]. Recently, analyses of Android permissions have been applied to generate various executing contexts for testing mobile apps such as in [4]. However, the authors selected little permissions that are related to communication with the environment for consideration. Therefore, there is need for further contribution.

4. PROPOSED APPROACH

The proposed approach aims to generate test cases from mobile app by combining both the context events and the GUI events. We extended the approach by Song et al. [4] for identifying context events from app's permission. Their analysis is restricted on app's manifest.xml file without analysing the source code. Therefore, the amount of information generated from the analysis is not comprehensive. Considering GUI analysis in combination with app permissions analysis can generate comprehensive information about events supported by an application.

Given an APK file the permissions and GUI widgets are extracted separately which are used to extract the context and user events as illustrated in figure 1. Details of the events extraction is discussed in subsequent sections.

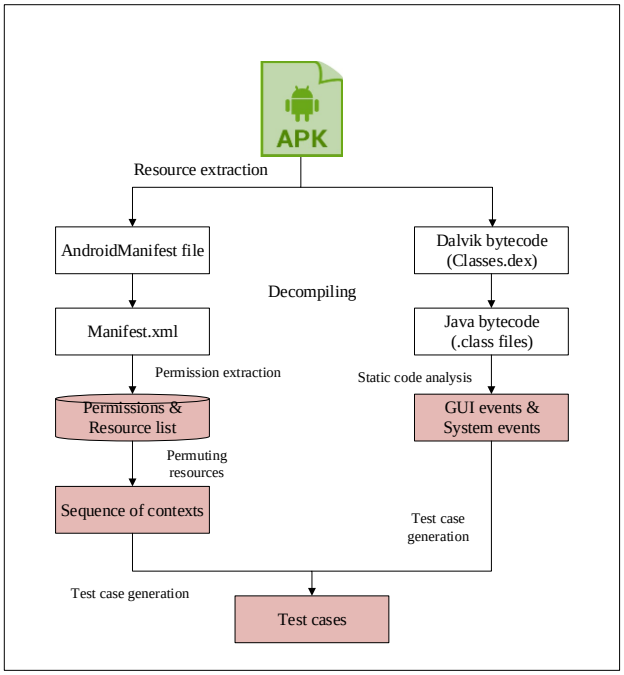

Figure 1. Illustration of the proposed approach

4.1 Application's Permission Extraction

The AndroidManifest.xml comprises all the permissions and resources that a mobile app may potentially use to run. The context events usually occur from the resources used by a mobile app. By analysing the AndroidManifest file and identifying permissions, the resources can be detected and subsequently the context events triggered by the resources.

The application's manifest.xml file is contained in the application package file (APK). An APK is a compressed file format which contains various resources that comprises an Android application. It comprises the program code (.dex file), assets, manifest file, resources and certificate. To extract app's permission, the APK file is first de-compressed to extract the resources. The xml file that contains all permissions assigned to an application is decoded using apktool's decode function to enable reading the information of permission. All resources that an application requires to run are declared with a list of the permissions in the manifest file. The resources protected by permissions can be accessed through the Android API and other classes in the phone.

In this research, an algorithm is proposed to extracts all the permissions by an application. The proposed algorithm receives as input the manifest.xml file and output the list of permission assigned to an application. The procedure starts in line 1 of the algorithm and declares the permissions and type of permissions used by an application in line 2 and 3. Then it creates a list of all the permissions in line 4. The algorithm checks if the list is empty in line 5, then it proceeds to find the permissions and type in line 7-8 iteratively. Finally, it extracts the permissions type and returns the list in line 9-10 of the algorithm.

Algorithm 1 described the extraction procedure

Algorithm 1

Input:Manifest.xml

Output:List of permissions

```
1   Procedure PermissionExtraction;
2   Up ← Uses-Permission;
3   Pt ← Permission Type;
4   Lp ← createListofPermission ();
5   If Lp is empty then
6       Lp ← findPermission (up, pt);
7       up ← getNextPermission;
8       pt ← getPermissionType;
9       Lp ← Permission (Up, Pt);
10  return Lp;
```

The permission list consists of permission name and the related resources as shown in table 1. To generate various scenarios of the executing context of an app, we applied combination method on the resource conditions. For all resources that have links, the candidate states can be combined to generate the set of executing context/scenarios. This information is subsequently used to generate test cases that are capable of testing the context events. We generated a list of permission and the resources that are related to the permissions using the list obtained from the manifest file as shown in table 1. The permissions assigned to an application are declared in a <uses permission>tag.

4.2 Static Analysis

The goal of the static code analysis is to identify all the events (GUI and system) supported by an application. Static analysis is performed on the bytecode of a mobile app using GATOR, a static program analysis toolkit for Android [33]. As explained in previous section, Android apps are released to the store as .apk files, as a result the bytecode is not available. We first extract the dex executable (classes.dex) file from the resources. The classes.dex file is decompiled using apktool (included as part of GATOR) to extract the java bytecode. The static analysed is applied to the bytecode where the GUI with their event handlers and the callback methods are analysed to identify the GUI and system events. The result of the analysis is represented as windows transition graph (WTG). WTG comprises nodes and edges representing app's window/activity and events respectively. The window/activity comprises GUI and their properties which can be used to trigger an event on the app.

As described in section 3.1, the handling of system events in Android is by invoking the Intent messaging system. The static analysis employed performs Intent tracking to identify the system events generated by the device hardware. The system events are other forms of context events [25] that can affect the behavior of an app. The WTG consists of events from both the GUI information and information extracted from Intent analysis. We analyse the WTG to extract information of the events with their trigger conditions. This is used to generate test cases for testing an application. However, due to the nature of static analysis, it only extracts the event based on static nature of the resource but it does not identify the dynamic behaviors of the resources. This is the main aim of applying combination on the resources to generate different scenario of events. The information is manually added to generate test cases.

5. EXPERIMENTAL EVALUATION

To evaluate the proposed approach, we applied it to analyse and test real-word mobile apps. The experimental evaluation includes 3 stages: selecting the subjects, analyzing and designing test data, executing test and reading the results. The goal of the experiment is to answer the following question. Does the approach offer a reasonable coverage of context events? Two open source android apps: Beem and SubsonicMusicStreamer were selected to be tested and their characteristics are summarized in Table 2.

Beem is a free and open-source app which is used to provide a full featured and easy to use Jabber (XMPP) client on Android. It has a 3.6 star rating by 824 reviews with a download of 50,000 – 100,000 on the Google play store. Subsonic music streamer is a free media player for music and video that virtually supports all audio formats. It can also be used as a remote control for music on a server. This application is selected because it was used to validate several testing techniques such as [4]. It has a 4.3 star rating by 6,433 reviews with a download of 100,000 – 500,000 on the Google play store as of December, 2017.

Table 2. Characteristics of the Tested Apps

App	LOC	Class	Method	Category
Beem	21179	227	1640	Communication
Subsonic	17779	702	4602	Health and Fitness

We applied our proposed approach to analyse, identify events, generate test cases and ran test on the selected subjects. Figure 2 summarises results of code coverage obtained for all the subjects.

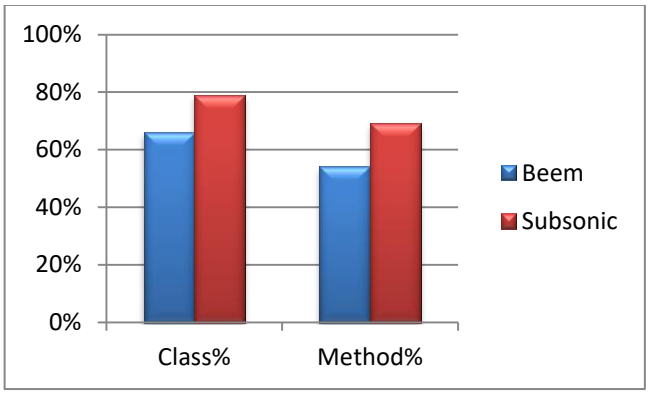

Figure 2. Code Coverage Result

As shown in figure 2 the class coverage and method coverage achieved by our approach on Beem app is 66% and 54% respectively. While the class coverage and method coverage for subsonic is 79% and 69% respectively.

Table 1 List of Permission with their Description and Related Sources

Permission	Permission Description	Resources	State
Access_Fine_Location	Access precise location	Wifi, Gps, radio	on/off
Access_Network_State	Able to access networks	Wifi, gps radio	on/off
Access_Course_Location	Access approximate location	Wifi, gps, radio	on/off
Write_External_ Storage	Write to external storage	Sd card	free/full
Use_Finger_Print	Use fingerprint hardware	Lcd, camera	on/off
Internet	Open network sockets	Wifi, radio	on/off
Camera	Access the camera device	Flash light, SDcard	on/off, full/free
Vibrate	Access to vibrator	Vibrator	on/off
Read_External_Storage	Read from external storage	SD card	Full/free
Receive Boot Complete	listen/reverie the BOOT_COMPLETED action after system finish booting	SD card	Full/free

6. CONCLUSION AND FUTURE WORK

This paper proposes an approach for testing mobile apps considering both GUI and external context events. We analysed app's permission obtained from manifext.xml file to identify context events and performed GUI analysis to generate the sequence of GUI events and system events through Intent analysis. We applied our approach on selected mobile apps and discussed the results. The coverage result showed that our approach had 54%-79% coverage across the two selected applications.

For future work, we plan to include more subjects in the evaluation and perform more experiments such fault detection to evaluate fault detection ability of our proposed approach.

7. ACKNOWLEDGMENTS

We would like to acknowledge the support from UTHM in undertaking the research under the Graduate Research Assistant for Postgraduate Research Grants (GPPS), FRGS Vot 1610, Universiti Tun Hussein Onn Malaysia.

8. REFERENCES

[1] Chan, P.P.K. and S. Wen-Kai. *Static detection of Android malware by using permissions and API calls*. in *2014 International Conference on Machine Learning and Cybernetics*. 2014.

[2] Choi, S., et al., *API Tracing Tool for Android-Based Mobile Devices.* International Journal of Information and Education Technology, 2015. **5**(6): p. 460.

[3] Dehlinger, J. and J. Dixon. *Mobile application software engineering: Challenges and research directions*. in *Workshop on Mobile Software Engineering*. 2011.

[4] Song, K., et al. *Generating various contexts from permissions for testing Android applications*. in *International Conference on Software Engineering and Knowledge Engineering, SEKE*. 2015. Pittsburgh, United States.

[5] Felt, A.P., et al. *Android permissions demystified*. in *Proceedings of the 18th ACM conference on Computer and communications security*. 2011. ACM.

[6] Amalfitano, D., et al. *Considering context events in event-based testing of mobile applications*. in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. 2013. Luxembourg: IEEE.

[7] Muccini, H., A. Di Francesco, and P. Esposito. *Software testing of mobile applications: Challenges and future research directions*. in *Automation of Software Test (AST), 2012 7th International Workshop on*. 2012. IEEE.

[8] Amalfitano, D., A.R. Fasolino, and P. Tramontana. *A gui crawling-based technique for android mobile application testing*. in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. 2011. IEEE.

[9] Nguyen, B.N., et al., *GUITAR: an innovative tool for automated testing of GUI-driven software.* Automated Software Engineering, 2014. **21**(1): p. 65-105.

[10] Liu, Z., X. Gao, and X. Long. *Adaptive random testing of mobile application*. in *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*. 2010. IEEE.

[11] Morgado, I.C., A.C. Paiva, and J.P. Faria. *Automated pattern-based testing of mobile applications*. in *Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the Quality of Information and Communications Technology*. 2014. Guimaraes, Portugal: IEEE.

[12] Méndez-Porras, A., C. Quesada-López, and M. Jenkins. *Automated testing of mobile applications: a systematic map and review*. in *XVIII Ibero-American Conference on Software Engineering, Lima-Peru*. 2015.

[13] Griebe, T. and V. Gruhn. *A model-based approach to test automation for context-aware mobile applications*. in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. 2014. ACM.

[14] Adamsen, C.Q., G. Mezzetti, and A. Møller. *Systematic execution of android test suites in adverse conditions*. in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 2015. ACM.

[15] Amalfitano, D., et al. *Using GUI ripping for automated testing of Android applications*. in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 2012. ACM.

[16] Salihu, I.A. and R. Ibrahim. *Systematic Exploration of Android Apps' Events for Automated Testing*. in *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*. 2016. ACM.

[17] Mirzaei, N., et al. *Reducing combinatorics in GUI testing of android applications*. in *Proceedings of the 38th International Conference on Software Engineering*. 2016. ACM.

[18] Amalfitano, D., et al., *MobiGUITAR: Automated model-based testing of mobile apps.* IEEE Software, 2015. **32**(5): p. 53-59.

[19] Hu, C. and I. Neamtiu. *Automating GUI testing for Android applications*. in *Proceedings of the 6th International Workshop on Automation of Software Test*. 2011. ACM.

[20] Yu, S. and S. Takada. *External event-based test cases for mobile application*. in *Proceedings of the Eighth International Conference on Computer Science & Software Engineering*. 2015. ACM.

[21] Liu, C.H., et al. *Capture-replay testing for Android applications*. in *Computer, Consumer and Control (IS3C), 2014 International Symposium on*. 2014. IEEE.

[22] Salihu, I.A., R. Ibrahim, and A. Mustapha, *A Hybrid Approach for Reverse Engineering GUI Model from Android Apps for Automated Testing.* Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 2017. **9**(3-3): p. 45-49.

[23] Chen, T.Y., et al., *Adaptive random testing: The art of test case diversity.* Journal of Systems and Software, 2010. **83**(1): p. 60-66.

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

[25] Yu, S. and S. Takada. *Mobile application test case generation focusing on external events*. in *Proceedings of the 1st International Workshop on Mobile Development*. 2016. ACM.

[26] Machiry, A., R. Tahiliani, and M. Naik. *Dynodroid: An input generation system for android apps*. in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 2013. ACM.

[27] Vieira, V., K. Holl, and M. Hassel. *A context simulator as testing support for mobile apps*. in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015. ACM.

[28] Wei, X., et al. *Permission evolution in the android ecosystem*. in *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012. ACM.

[29] Android permissions, *https://developer.android.com/guide/topics/manifest/manifest-intro.html.*

[30] Johnson, R., et al. *Analysis of android applications' permissions*. in *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*. 2012. IEEE.

[31] Zhang, Y., et al. *Vetting undesirable behaviors in android apps with permission use analysis*. in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013. ACM.

[32] Chan, P.P. and W.-K. Song. *Static detection of Android malware by using permissions and API calls*. in *Machine Learning and Cybernetics (ICMLC), 2014 International Conference on*. 2014. IEEE.

[33] *GATOR: Program Analysis Toolkit For Android*. http://web.cse.ohio-state.edu/presto/software/gator/.