A Scoping Review on Automated Software Testing with Special Reference to Android Based Mobile Application Testing

Fathima Naja Musthafa[#1], Syeda Mansur[2], Andika Wibawanto[3] and Owais Qureshi[4]

***Abstract*— Despite all the techniques practiced for ensuring the quality of a software product, software testing is being the widely accepted practice. With the explosive evolution and the usage of mobile application, new developments in the process of software testing are introduced too to acquire market presence in mobile application development by introducing high quality products. As of this the introduction of automated tools for testing has gained attention in the last few years. Though the topic of automation in software testing has been there for a while, introduction of new tools and techniques has gained attention recently. Hence, this research work focuses on investigating and analyzing the current trends on automated testing of mobile application by choosing the android platform as a case study. With the aim of fact finding, a systematic literature review was carried out on existing studies which were retrieved from different databases by exploring the electronic search space. It discusses the points based on the chosen research questions by referring the papers cited. The topics discussed in this review article includes the facts related to why and how automated testing on mobile application, the tools and techniques used and the challenges on it. This work also highlights why the focus has been concentrated on the mobile application testing rather than generally highlighting the importance of automated software testing. As a conclusion the paper proposes some good practices on the topic based on existing literature reviewed and referred throughout the study.**

***Keywords*— Automated Testing Techniques, Mobile Application Testing Tools, Quality Assurance, Software Testing**

I. INTRODUCTION

Despite the works done by researchers and practitioners about the numerous techniques for software quality assurance, it is widely accepted that software testing is the most practiced approach for evaluating and assessing the quality of a software product [50]. The main goal of this paper is to provide the insights from successful research works carried out in software testing and testing techniques for mobile application which appears to be the most significant points relevant to the topic.

Correspondence: F. N. Musthafa [#1] (E-mail: mmfnaja@gmail.com)
Received: 28-12-2020 Revised:17-05-2021 Accepted: 24-05-2021
F. N. Musthafa[#1], S. Mansur[2], A. Wibawanto[3] and O. Qureshi[4] are form University of Malaya, Malaysia. (mmfnaja@gmail.com, smansur.irene@gmail.com ,andika.wibawanto@gmail.com ,umerkhattab42@gmail.com)

This paper is an extended version of the paper "*Automated Software Testing on Mobile Applications: A Review with Special Focus on Importance, Tools and Challenges in Android Platform*" presented at the ICTer Conference (2020)

DOI: http://doi.org/10.4038/icter.v14i3.7227

Li and his co-authors, in their work published in 2014 state that the main objectives of software testing are:

1. A test is carried out to demonstrate the errors that is present in a product.

2. A well-defined testing approach has the higher chance of discovering errors that exist.

3. A successful test operation should always discover any future faults and regression failures.

The common term used in literature as 'mobile testing' refers to various testing strategies like testing mobile devices, testing mobile applications and mobile web applications testing [22]. Thus, the term 'mobile application testing' in this paper refers to testing mobile applications that run on mobile platforms with the use of popular testing methods and tools that ensure quality in behaviors and functions as well as features like usability, security, connectivity and so forth. Various work in the literature highlights the fact that mobile application testing is much different from the conventional software testing as it has unique requirements which includes device compatibility of the application with different mobile devices with ranging screen sizes to UI lags [35]. Apart from that, since mobile applications are developed to run on mobile devices that operates on different operating systems, having different size and computing power resources [22], the way of testing those also must be to the standard that differs from normal conventional software products. Hence, this paper clearly highlights the importance of testing mobile with the support of literature in the second part of section two 'critical evaluation of literature'.

According to Cap Gemini Quality Report [1], the barriers to testing mobile application have moved from tools to methods; 56% of companies do not possess the right testing process/method, 52% do not have the devices instantly available, 48% do not have test experts, 38% do not have in-house testing environment, 37% do not possess the right testing tools, and 33% do not receive enough time to test. However, the data shows that mobile testing rose rapidly in 2013 compared to 2012 where statistics prove that 55% of organizations implemented new methods and tools to test functionality, performance, and security of mobile applications and devices in contrast to 33% in 2012. The rise in percentage is optimistic.

Based on the strategies used to carry out the testing process, the automated software testing for mobile application is classified under different categories. Although various techniques used differ in the approaches used, these testing does not fail to accommodate the concept of

automation. This paper gives a detailed insight of some of such techniques used and the tools utilized in addressing the features of those techniques under the third topic of the second section.

Despite the developments and research in software testing for mobile applications, the challenges faced continues in relevant to test environment and standards, modeling, and coverage criteria [22].

This paper is organized in the form of sections. Section one states about the content of the sections in this paper as well as highlight the key points explained in this review followed by the second section that discusses the research methodology adopted for this study. The third section discusses the findings and results based on the identified research questions, that explains about the key phrases found to be relevant to the topic and they are well explained with the support of literature referred in there. The final section is the conclusion, and it discusses the key points and important conclusions arrived during this literature review and provide information about major recommendations on these topics. Reference list follows the conclusion part, and it lists all the references used for this literature review work.

II. RESEARCH METHOD

This study adopts a systematic literature review process proposed by Kitchenham et al. on how to perform systematic literature review in software engineering fields [2]. The subsequent parts of this manuscript shows how the methodology has been adopted in conducting this systematic literature review.

A. Research Questions

There are four main research questions in this review and the discussion part mainly adhere to the focus on these research questions.

1. What does automation in software testing refers to?
2. What is the importance of automated mobile application testing?
3. What tools and techniques are used in automated software testing of mobile applications with respect to android platforms?
4. What are the major challenges in automated software testing on mobile applications?

B. Search Strategy

The search for study was conducted in the electronic search space. Electronic databases were explored with the target of research questions based on key word search approach. To identify the suitable literature, inclusion and exclusion criteria was used among the search results. The researchers also used some screenings on the process of identifying the relevant literature to remove redundancies and irrelevant studies with mutual consents.

C. Information Sources

Popular scientific databases were chosen to conduct the electronic search and retrieve the relevant literature for this review. The databases include IEEE Xplore, ACM digital library and ScienceDirect. Additional records were also found via google scholar.

D. Search Terms

The scope of the study being little broader without fixed taxonomy, a range of search strings were used as keywords across the electronic search space. The keywords defined were used with Boolean combinations such as “and” and “or” with the aim of minimizing the irrelevant results. The keywords are “Automated testing”, “Mobile application”, “Testing tools”, “Quality Assurance”, “Software Testing”, “Test automation”, “Testing challenges”, “Testing techniques”, “Android platform”, “Mobile Devices” and “Test strategies”.

E. Inclusion and exclusion criteria

The literature selection for the study opt to be with the following characters.

1. The study must be published within the timeframe of 2000 to 2019.
2. The study must be in English
3. The study must include relevant information of mobile application testing covering the scope of the study.

Studies which were excluded were based on the following concerns.

1. If the study does not fulfill the inclusion criteria
2. If the study does not technically prove to be rich in content.

III. LITERATURE REVIEW AND KEY FINDINGS

This section highlights the facts adopted from the reviewed literature that adheres to the research questions identified and given in the section two of this paper. Although the main focus was to investigate the current state-of-the-art facts based on the relevant research questions and the key topic, this section tends to summarize the findings in the form of structured way as to clearly relate the facts identified from different literature so as to provide the connectivity between the findings. Also, the results are based on literature and articles published between 2000 and 2019, thus giving importance to the recent developments in the topic. Though the articles retrieved from the internet are verified according to the source origin, it was double checked for the authenticity of the content. The consequent sub sections of the third portion of this paper provide detailed discussion on the research questions thereafter.

A. Software Test Automation

Software Testing is one of the important phase of any software development and has been widely used in the industry as a quality assurance technique for evaluating the specification, design and source code [23], [7]. Since the software is designed in a way that is more complex, the need of testing complex software becomes the important phase of any software development. Hence, the importance of testing should not be underestimated [60]. In fact, software testing is a part of any software development and plays a major role in the cost factor of any software [23]. Software testing is expensive and labor intensive. According to literatures, Software testing process covers up to 50% of software development costs and it is even more for safety-critical applications [6].

The main objective of a software testing is ensuring a quality software product [23]-[25], [50], [27] at the end of a development phase before it is put into deployment. It does not mean that software testing is carried out at the end of an Software Development Life Cycle (SDLC). But it may be performed at any stage wherever necessary and it totally depends on the project and the model of SDLC used for the development.

In the process of software testing, an outcome of a software development process is evaluated for the overall functionality and behaviors using a set of test cases - whether it satisfies the specification requirements or shows any behavior of fault in the software. Although the concept of software testing is stated to be used for demonstrating the absence of errors [35], [27] in any software product, testing is always defined to be finding errors as much as possible as it improves the assurance that the software being tested is much reliable. Thus, a set of test plans are executed to find out that. The way in which these test plans are executed, divides the software testing process in to two major categories like manual and automatic testing.

In manual testing, a tester carries out a written set of test plans which contains the test cases [23]. Here a tester manually executes the program to check for each test case. As this is the approach of manual testing, the automated testing is automating these test activities and the whole test cases are carried out automatically.

Garousi and Mäntylä describe the automation testing in their work published in 2016 as the “use of special software (separate from the software being tested) to control the execution of tests and the com-parison of actual outcomes with predicted outcomes” [23]. Key points that these authors trying to mention are “special software” and “control the execution’. By this, the authors mean that the testing process uses a software other than the software needed to be tested and the whole process is automated. Although various researchers define the term automation testing in their own style of wording, none of them have missed to portray the same concept.

In every test activity, it is always essential to find out why an approach is selected. Since software testing is one of the major phases in any software development, it is labour intensive and expensive. According to a literature, it is stated that testing takes up to 50% of the total cost of any software development. It is sometimes even more than that according to some literature. As this is the fact regarding testing cost, it is essential to manage it and that is the goal of automation testing.

Another goal of automation testing according to literatures is, minimizing human error [30]. Mistakes, made by human beings, be-come errors which tend to become faults and failures. Another goal is making regression testing easier [30], meaning, that when automation testing is executed to find the errors in any software testing, it makes the process of finding any consequences of any patch works done during a bug fix. Thus, this will ease the problem of overcoming any possible future errors caused by a bug fix. Software testing phase on any software development process tend to start along with the beginning of the development in order to avoid the complexities in testing at later stages. Although the approach implemented for testing fully depends on the basis of project requirements and the model of software development used, it is mandatory to thoroughly study which approach is to be used for the purpose of testing in order to avoid future errors and failures in testing. Thus, it is essential to identify when and what is to be tested using a particular testing strategy. A test process comprises several steps, beginning from planning, test specification, executing up to reporting. Each of these steps could be carried out using various approaches. Apart from using automation testing in the test execution process it could be used in various stages for various purposes too. Thus, the potential use of automation in various testing stages can be test case design, scripting, evaluation, and test result reporting [23], [27], [30], [32]. Based on these, an overview of this is summarized in figure 1.

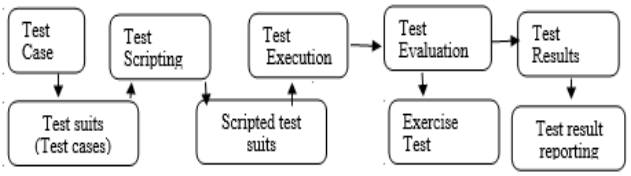

Figure 1: An overview of automation across the software testing process

B. Importance of Automated Mobile Application Testing

Recent years have proven to be a great revolution for mobile industry and market due to its extensive support and sophisticated tasks being performed, rather than just simple operations as a decade ago [32], [33]. The mobile has been used by many giant tech industries and companies which are not directly related to technology. Mobiles are being used as a common platform to manage tasks for everyday life and activities. Since its involvement has become a norm in our lives for every mere and essential task, any breakdown/unexpected behavior of any sophisticated mobile application can result in a great disaster for industries and enterprises.

Users’ unpleasant experience (crashing, bugs etc.), while exploring mobile app, can consequently prevent the users from reusing the app. Nearly 48% of users will not try the application again based on one survey [14]. This can lead to lower downloads, thus reduced revenues. Therefore, to avoid such consequences after all the time, energy and money invested on any mobile app, various Software Testing on

mobile application is certainly necessary. Meeting graphical interface sketches, functional requirements and flow of app can be achieved and may attract users. Ensuring high quality is essential than anything else.

The audience of mobile application is escalating immensely as mobile devices continue to be used and seen everywhere. MarketWatch's research states that, 14% of online purchases are made through mobile platforms and will continue to grow gradually in upcoming future. One of the statements from PayPal's senior director of global initiatives, Anuj Nayar, to MarketWatch states that "we've seen our mobile growth rise from less than one percent of our payment volume in 2010 to more than 20 percent in 2014". This notable new height in this area shows more and more businesses are cashing-in on.

Needless to say, making sure an app is working correctly is essential. The same hard work that is required for product concept and building a business, is also necessarily to be done with quality control and testing for mobile applications, and that kind of testing is not something that can be done in-house. It can only be achieved by using professional mobile testers' skills, who can identify issues before it affects the end-user which gets them frustrated, as well as architect ways to fix them before the application is rolled-out.

The following comparison reflects the idea of why mobile application is foremost important than web and desktop platform. It also provides an insight of all possible aspects which explains why mobile platform application is complex, time consuming and detailed in comparison to its counterparts.

TABLE I. IMPORTANCE OF MOBILE APP TESTING OVER WEB / DESKTOP SYSTEM TESTING WITH COMPARATIVE SUMMARY

Criteria	**Mobile**	**Web/ Desktop**
Frequency of release	Release Software updates quite frequently for the improvement of devices, security and UI lags, etc. This effect mobile behavior in a way that old compatible apps stop working. Thus, the testing team needs to be cautious.	Web and Desktop versions are released not so often, hardly twice or once in a year or two.
Usage	Mobile apps have large adaption from mid – high level enterprises and being used for a general purpose which makes it complex enough to support all genre of applications supporting up to 13000 devices as per google play console [19].	Desktop applications are mostly used by big enterprises where there is less variety of desktop machines to support. For web-app, only concern is on which cloud or another server the web-app needs to be deployed.
Communication link	Mobile applications are connected through sophisticated interlinked bridge called Restful APIs or web services which make the transaction happen through a mobile meaningful format known as JSON [56].	Web application is hosted on the same server where the database is deployed. Thus, the chances for being vulnerable in terms of security, non-availability of data is less than its counterpart mobile application [21].
Development life cycle	Built with a complex life cycle to handle all kinds of unexpected, interrupted behaviors in more intelligent way.	No such life cycle due to not being developed with an intention for being a personal platform rather being originally used by everyone on same hosted server serving millions of users.

Possible outcomes, when any app is not tested with the context of mobile lifecycle, may lead to great consequences such as, mobile being used with a purpose of multi-tasking device. It drives the concept of Background and Foreground app [18]. However, Backgrounded apps brought to the foreground will often crash if state is not persisted properly. States can be seen in Figure 2 – Android Lifecycle (For instance, OnStart/OnStop and viewWillAppear/viewWillDisappear for Android.

1. The testers should also be aware that Mobile's intelligent algorithm destroys app whenever it deems necessary for more memory allocation. If OnDestroy state is not correctly defined it might lead to unexpected behavior or loss of user's data on next round of using app.

2. Mobile's platform also forces the developers to build the cache controller to load heavy data for avoiding misinformation or non-availability of data [37].

3. There are often multiple ways that a hook can be called, and the testers need to be aware of the differences in certain situations.

4. As what was mentioned during comparison in table 1, often updates/patches may affect the system's overall flow in older android system. No guarantee of OnStop state being called on request is given even by developers. Thus, reviewing official documentations is also a way to cope with unforeseen bugs by mobile testers.

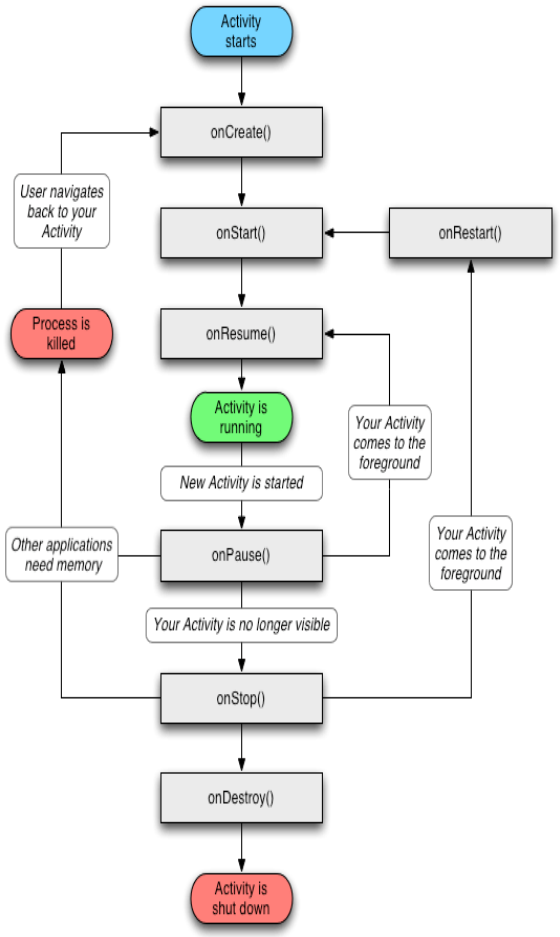

Figure 2: Basic Android Activity Lifecycle

As we have noticed in our mobile phones, even though our phone's screen is off or partially idle state/sleep mode, it still popups and alerts messages/notification. For instance, WhatsApp/messenger text messages notification. This entire process runs in a background called a background thread/service which is mostly used in Real-Time service intact application as it can be noticed in figure 3 below, most top 10 crashed application of 2017 list were those which uses background threads almost for every task.

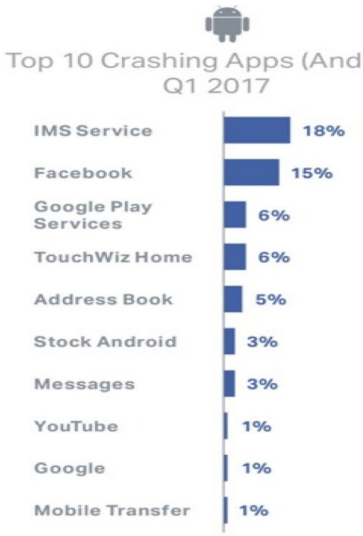

Figure 3: Survey results of top 10 crashing apps as given in [18]

In addition to background-running apps, some critical user conditions to be tested against include:

1. Geographical location.
2. Device and operating systems commonly used in these geographies.
3. Most-used mobile apps running in the background.
4. Network conditions.
5. Interruptions occurrence while using app (calls, messages, other popups).

When testers mimic such experiences while testing the app, the recommended way to assess such application is to get familiar with the application's type along with real-time service providers.

C. *Tools and Techniques used in Automated Software Testing of Mobile Applicationsin Android Platform*

Based on recent survey by Stat Counter, Android operating system is the most popular operating system in the world [47]. The test input generation tools for mobile application usually target mobile apps developer with primary goals to detect existing faults in mobile apps or to maximize the code coverage. The source code of the app must be open source in order to allow the tool to do checking, and after checking is done the mobile apps developer then can catch the possible errors and fix them. The functional defect is not the main problem for apps developer because they can do testing manually, the most important concerns are portability, malware and energy issues that can be effectively detected by executing the code.

Most of the time the mobile apps are in the idle state waiting user input such as clicks, scrolls, or system event, such as notification, SMS, or GPS location update. Application also may need input from the users by input certain value into widget, selecting from a list, and so on. Because mobile application is event-driven, testing tools will treat an input from the user as an event or break them into sequences of event that model user action or model user input. The sequences of events and the inputs can be generated by random value or can follow a systematic approach. A systematic approach, usually the model of the application, is guided by the process to limit the search space. These models can be manually, statically or dynamically built.

The capture and replay technique or model-driven technique are state of the art to manage traditional event-based system. In the capture and replay technique [3], [4], tester first do manual testing by recording his interaction with the GUI then the recording will be replayed during testing automation. In model-driven technique [58, 41] mobile application model to be created first before automation testing can be done. Both techniques require tester involvement, thus may not detect corner case that human testers are unaware of. Another technique that does not need manual tester involvement is by extracting directed graph model from the GUI with crawling technique like in web development [3,42] test sequences is produced by those graphs, but still may fail to identify a system that can be explored with.

The android software development kit is already included with powerful testing framework [11]. The android testing framework is an extension of Junit framework with addition of tool to test specific Android application. The addition is to address fundamental issue with mobile application

development like Android views, Activity, Content Provider, and specific set of Assertion classes designed for them.

There are many tools available outside of default Android testing tool. Robotium [52] also build on JUnit framework. Robotium uses GUI assertions like web application testing with Selenium framework. The Selenium framework is very popular and simpler to write tests with and mostly used for Black-box testing, and is very useful to do functional testing, system testing, and acceptance testing.

UI Automator [57] does not use JUnit but provides same functionality for the test engineers to build GUI tests like clicking buttons, text input, scrolling and swiping. The Uiautomator has special abilities to check the state of application before and after user actions in GUI; this can be useful for Black-box testing of apps through GUI. It also supports GUI assertions. Monkey Runner [9] has Android Emulator that can be controlled from outside of Android code. Monkey Runner can provide screenshots and is very useful for Regression testing by comparing screenshot with functional testing. Espreso [12] is the latest Android test automation framework by Google. It is a more advanced generation and builds on top of Monkey Runner. It has similar functionality but is more reliable. The subset of espresso is Espresso Test Recorder which can record the interaction with device and do assertion to verify UI element. This recording can be rerun in the future. Robolectric [52] uses java reflection API at runtime and use shadow class to test on real device outside of emulator. This tool also has the ability to run the test directly accessing Android libraries file with Java reflection API. It replaces the body of Android API methods at runtime using java reflection.

The type of mobile app testing techniques and information on the tools adopting these techniques to test android apps are summarized as follows.

1. Radom Techniques

Based on the study of Choudhary et al. [17] random testing technique is the best automated testing for Android app. The Android Monkey [8], one of the tools they studied, is the best performance tools available for test input generation tools.

Android has characteristic of event need to be initiated by user or system event by Android framework itself. Usually system event can be triggered by specific condition. As a result of this behaviour random testing is not very efficient. Most of random testing technique for Android such as [40], [8], focus on generating only GUI events.

Android Monkey is a part of the Android developer's toolkit and is widely used by both developers and app market managers. It uses brute-force mechanism that generate pseudo-random streams of user event such as clicks, touch, gestures in a random. It is Monkey that fires off both GUI and system events based on the number of events that are specified by the tester and utilizes a completely random strategy [8].

Dynodroid also uses random values and sequences of events, but it has added few heuristic approaches to improve Android Monkey's performance [40]. One of the approaches is checking android manifest file to generate only relevant system events for the application. It also keeps the track histories of the type and number of events used and not randomly generated next event but uses a least recently used algorithm. The tester can also manually enter specific values for specific input text like text boxes to make it least random.

There is also another group of random testing techniques [54], which focuses on testing inter-application communications by randomly generating values for Intents (Intent fuzzing). Intent fuzzers mainly serves the purpose of generating invalid intents to test application robustness and to reveal vulnerabilities by generating malicious random content. Several other approaches are built on random testing techniques. Amalfitano et al. [4] presentet a GUI crawling-based approach like in web application testing with Selenium framework that uses random inputs completely to generate unique test cases. Hu and Neamtiu [29] describe a random approach for generating GUI tests that use the Android Monkey to execute. Random testing techniques are very efficient to generate events, but it is not suitable for generating specific input. They also produce redundant events that are already covered in previous cycles.

2. Model-Based Techniques

Web-based testing application give inspiration for Android testing. They follow same technique in an event-based system to systematically generate sequences of events that resemble the behaviour of the mobile application. The tools discussed below use static and dynamic analysis technique to generate machine state by capturing the activities of the application in the transition of events.

MobiGUITAR [4] builds a model of the application by dynamically exploring an App GUI with GUI ripping technique. It builds on top of GUITAR [50]. The model then is traversed by a depth-first search strategy to generate test cases. When the tool cannot detect new states during traversing then it can be restarted. MobiGUITAR also can use random strategy or tester can manually input constant values during exploration.

ORBIT [60] analyses the source code and manifests file to identify relevant UI events. It also statically analyses source code to identify state transition between activities. This technique is called grey-box model because it analyses not only the GUI but also the source code.

While ORBIT uses static analysis SwiftHand [16] uses dynamic analysis and machine learning to organize state model of the app during testing. The machine learning is used to visit unexplored states of the application. The model is refined dynamically during the execution of the app using the generated inputs. The main focus of SwiftHand is to optimize the exploration strategy in order to minimize the restarts of the app during the exploration.

A3E [13] also uses static analysis technique for building an app model for automated exploration of an app's activities. A depth-first search strategy is used for reaching a certain state in model. This technique is important for construction of model testing.

PUMA [27] uses dynamic analysis to build the model. The goal of this tool is more to provide infrastructure for dynamic analysis of application. It is built on top of Uiautomator [57]. Instead of reinventing the wheels PUMA uses Monkey's exploration strategy but it provides a framework that can be extended to implement any exploration strategies.

Most of the tools above focus on construction of models for testing that are covered using a depth-first search strategy for the generation of event sequences. Model based technique are useful for complex application that has infinite state and cannot be explored using random technique.

3. Record and Reply Techniques

Monkey Recorder [9,10] and RERAN [26] implement record and replay techniques for Android apps. Monkey Recoder allows testers to record a script for GUI events of an application on the device and the recording can be saved and rerun in the future. As of now Monkey Recorder only collects click, swipe, and text-input events.

RERAN, on the other hand, logs the event system commands of the Android operating system to generate low-level event traces. Because it is low level event it is dependent on the hardware like screen size and cannot be rerun in other devices. These scripts are analyzed and turned into runnable scripts. RERAN replays the recorded script [26].

Record and replay technique can be useful for stress testing and regression testing, but the scripts need to be generated manually. Because of this they are usually biased towards only certain features and do not capture the behavior of the app completely. These techniques can only replay what is recorded and do not consider other combinations of events for replay.

TABLE II. SUMMARY OF TECHNIQUES WITH THEIR ADVANTAGES AND DISADVANTAGES AND TOOLS ADOPTING THE TECHNIQUES

Technique	Advantage	Disadvantage	Tools	Reference
Random	Efficiently generates events, Suitable for stress testing	Hardly generates specific inputs, Generate redundant events, No stopping criterion	Android Monkey [8], Dynodroid [40]	[17],[40],[8], [54], [4]
Model-based	More effective, Can reduce redundant events	Does not consider events that alter non GUI state	MAMBA, SSDA, MobiGUITAR, Orbit, SwiftHand, A3E, PUMA	[5],[57], [50], [60],[27], [68], [69]
Record and Replay	Useful for stress and regression testing	Test script are generated manually	Monkey Recorder, RERAN	[9],[10], [26], [67]

D. Challenges in Automated Software Testing on Mobile Applications

Several studies have been conducted by many researchers, on the challenges of mobile apps testing and its potential research possible targets [15]. All their studies come to some common major challenges they found. Some significant points noted were (1) Mobile applications are very different from traditional ones and thus different and specialized techniques are involved in the testing and (2) there are many challenges, most still with no optimum solution [37]. For instance, the randomness of the testing environment greatly manipulates the reliability, performance, security, and energy. In the following section, some of the major challenges are discussed.

1. Device Fragmentation

One of the major challenges of Software Testing is Device Frag-mentation [32], [4], [19], [32], [45], [1], [14], [33]. Variations in the hardware or O.S. components can cause mobile applications, while running on different devices, to behave differently where each application has its own unique business and data flow [1]. A study reported the existence of 1.800 hardware/O.S. different configurations as of 2012 considering the fact that (as of 2012) there were around 130 different mobile phones operating on Android, 7 versions of the OS, and presuming two firmware per device [49].

Mobile device fragmentation is a phenomenon that takes place when older version of an OS runs on a device, while newer versions are already in existence. There are several mobile OS available. An app performs differently in different platforms. A testers goal should be to provide a consistent user experience across platforms. Using a framework that supports multiple objects can help as it assists to isolate the functionality of a specific object, determining whether it needs an alter for other platforms or not. For instance, if an app has a selection menu that needs to present as a scrolling list for Android and a radio-button selection list for Windows Phone, a testing solution is required that supports multiple objects, to test both the scenarios [14]. According to Testing Experience Test devices – Fragmentation can be grouped into three categories [35]:

Group 1: Small devices having a small CPU, RAM and low resolution, older software versions and older browsers.

Group 2: Mid-range devices having an average CPU, RAM (<512 MB), good screen size and resolution, older software versions.

Group 3: High-end devices having a dual/quad-core CPU, RAM (>512 MB) and a high screen resolution, latest software versions.

Therefore, the following choices adds to the challenges when testing on varied combinations of devices with right combination of operating systems: whether to use manual testing or automated tools, in-house teams or outsourced partners, guided testing or exploratory testing, emulators and simulators or remote access [35],[15].

Due to compatibility issues, different user interfaces increase level of challenge. User's application experience is significantly affected by mobile devices network performance; where multiple network technologies may be supported by each mobile operator and unfamiliar or local networking standards may be used by some as well. To test mobile application in all these probable connected networks, travelling to every network operator is commanded which can be very costly and time consuming. Although this network challenge can be overcome by bypassing the lower layers of network to test the application via Internet on network by using device emulator and thus saving time and cost of travelling, bypassing cannot exactly imitate the effect and timing of network. Security is another aspect of the effectiveness and validity of the application; ensuring the application is secured and does not surpass user's private and sensitive data is thus mandatory. Significant hardware component in addition to its system (for example, GPS, telemetry, scanners etc.) presents a great challenge and since

mobile applications are used by different category of people stretching from zero IT background to top notch IT, the usability testing must involve a comprehensive range of scenarios taking into consideration in their own environments [47].

2. Connectivity

Apart from the hardware and software issues, the functionality of an application is also affected by the performance of carrier's network. The application is expected to work with 2G, 3G, 4G or 5G network, Infrared, Bluetooth, GPS, NFC (Near-Field Communication), WiMAX, low signal strength and different Wi-Fi speeds [1]. Some applications are even expected to work the same in no-network or offline condition as well with synchronization done [19], [14]. In addition, a single application can also be expected to sustain in multiple types of connectivity simultaneously [32]. Slow and fallacious wireless network connection having low bandwidth is found to be a common obstacle for mobile applications in many studies [35]. Network latency (time taken to transfer data) will be random when apps communicate over network boundaries. This results in unpredictable speeds in data transfer [14]. Gateways in a wireless network convey content more appropriate for specific devices while acting as data optimizers. Again, data optimization process may result in decreased performance for heavy traffic. Testing should establish the network traffic level at which the performance of the mobile application is influenced by gateway capacities [14].

3. Device Limitations

It may be unsuitable in some devices to interpret images locate elements on the screen resulting from the difference in display sizes across mobile devices and their various models. Limitations in processing speed, memory size (RAM, secondary storage), CPU power, power management dependencies, battery life dependencies, cumbersome input UI of mobile devices result in variations of application's performance across different types of devices. The display capability of mobile devices supports much less display resolution in comparison with desktops. Low resolution can degrade the quality of multimedia information displayed on the screen of a mobile device [62]. So, testing must guarantee that the application has the capacity to deliver optimum performance and usability for all anticipated configuration of the hardware and software involved. Mobile devices also have different application runtimes. Some of the runtimes commonly available in mobile devices are Binary Runtime Environment for Wireless (BREW), Java, and embedded visual basic runtime. Applications should be tested intensively for the variations particular to runtime only [14].

4. Input Interfaces

To input user data into a mobile application touch screen is mainly used. However, the device resource utilization affects the system response time to a touch, and it may become slow in certain contexts like entry level hardware, busy processor and so on. To validate the touchscreen performance under different such contexts (for instance, resources handlings, load from processor, memory and so on) and within different mobile devices, testing techniques have to be created.

Different context contributors may provide inputs to mobile apps as well, i.e., users, sensors (like noise, light, motion, image sensors) and connectivity devices (some examples have been mentioned earlier), inputs that vary from different as well as changing contexts the mobile device can step towards. All those devices may supply a combination of inputs starting from brightness, temperature, altitude, noise level, type of connectivity, bandwidth to even neighboring devices that vary, even unpredictably, subject to the environment and user activities [49],[35]. Validating whether the app is going to appropriately function on any environment and given any contextual influence is a conundrum and may result in combinatorial explosion.

5. Rapid Application Development (RAD) Methodology

In order to cater to the benefits of faster time to market, RAD environments are exploited for mobile application development. Since the introduction of RAD tools reduce the time taken for development, builds are presented for testing much earlier. RAD methodology thus enforces an implicit pressure on testers to reduce the testing cycle time, not compromising quality and coverage of course [14].

IV. DISCUSSION AND CONCLUSION

A thorough review of literature has been conducted to provide detailed discussion on the identified research questions. The study has given priority for the android platform to discuss certain points like tools and techniques. This has been done to specifically mention few points with example as there a number of mobile platforms in use these days and android is considerably one of the popular one. Though the reference is on android platform, the challenges and importance of mobile application testing when performing automated testing, as discussed under the A and D subsections of the third section is common to all the mobile application and certain points in the B and C subsections can also be taken into consideration for other platforms too. As the main idea behind this study was the target on the practitioners of automated testing, this study would definitely be useful for them to gain certain knowledge on the topic and the researchers in this field too would be benefited from the findings of this study. As of that, the authors wish to summarize the following to the practitioners of the automated software testing on mobile applications as the best practice to be adopted and these facts are based on the literature review carried out for this study.

Literature reveals that all the software testing techniques involve in ensuring a quality product, before the implementation of any such technique to evaluate a software product, the particular approach must be well studied as it may be the ideal approach for the testing process. Based on the way the testing process is carried out, testing is classified as automation and manual. Although the potential benefits of using automation techniques are higher compared to manual testing, it is always advisable to look in to when and what to auto-mate and whether the particular approach can well define the needs. Although from low scale to high-end scale business have adapted mobile as a source to accomplish their daily tasks professionally. But, today the enterprises/businesses are more concerned with application that scales automatically as per the data grow along with high availability of access across the globe. Its functionality, usability, and consistency, these all characteristics can be evaluated, by performing automation or manual testing for

high quality of application, end-to-end testing is also very important to ensure application downloadable effectively, works seamlessly, and no lag during screens transitive which makes mobile application testing different from other platforms but complex and lengthy for testers due its complex lifecycle and detailed process and understanding of back-ground threads and so. At the initial phase, practitioners could commence the automation testing using tool like Monkey because it is already included in standard Android developer toolkit and does not need any additional requirements. Monkey is also very popular and has support from Google, the owner of Android. No single strategy alone seems to be effective enough to cover all behaviors; a combination is more effective. Random technique can be used for stress testing, record and replay is suitable for regression testing, and for application that is complex and has many UI activities model-based can be considered. In terms of challenges as we can see challenges in mobile application are huge in number and complex in nature, it is either required to plan a test strategy that is mobile-specific, or else we may over-look crucial areas of testing like how network connectivity (or lack thereof) distresses an application, how screen resolution and orientation changes could spoil a user's whole experience, and whether our application accomplishes what users of a particular device have come to expect, or, we may opt for something like what Google, the big blue chip, is researching on, which is, modular phones. As an effort to come up with an approach that amalgamates most benefits of the other approaches, Google endeavors to introduce new modular phone. A modular phone with working user-interchangeable components can let the users to upgrade their mobile easily and efficiently as all main components are interchangeable via modules that click in and out; this can facilitate testing process as well.

Apart from the best practices suggested for the practitioners, the researchers in the field would be suggested for using this study for any domain specific research related to this field as the study gives detail specifications under the topics discussed. As the authors take this as the starting point for further research to be carried out on the compound testing strategies ideal for automated software testing for mobile application, the future research on this study could be expanded based on the software testing phases where automation can be implemented while automating the whole procedure of software testing too. Apart from this, the researchers in this field could also investigate the challenges specific to mobile application testing and whether it could be mitigated with automated tools. Also, as this scoping review mainly refer to the android platform, future studies are also encouraged on other platforms.

ACKNOWLEDGMENT

The authors of this paper work take this opportunity to thank all those from the Department of Software Engineering, University of Malaya, Malaysia who helped throughout this research work. The authors also complement and acknowledge the reviewers and the panel of the ICTer2020 conference for suggestions and ideas for improvements as the concept of this paper was presented as a poster at the ICTer 2020, Colombo, Sri Lanka.

REFERENCES

[1] Akour M., Ahmed A., Falah B., Bouriat S., Alemerien K. (2016), "Mobile Software Testing: Thoughts, Strategies, Challenges, and Experimental Study", Vol. 7, No. 6, 2016.

[2] Kitchenham, B., Pretorius, R., Budgen, D., Pearl Brereton, O., Turner, M., Niazi, M., & Linkman, S. (2010). Systematic literature reviews in software engineering – A tertiary study. Information And Software Technology, 52(8), 792-805. doi: 10.1016/j.infsof.2010.03.006

[3] Amalfitano D., Fasolino A., and Tramontana P. (2011), "A GUI crawling-based technique for android mobile application testing". Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, March 2011, pp. 252–261.

[4] Amalfitano D., Fasolino A. R., Tramontana P., De Carmine S., and Memon A. M. (2012), "Using GUI ripping for automated testing of android applications". Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ser. ASE 2012. Essen, Germany: ACM, 2012, pp. 258–261.

[5] Amalfitano D., Fasolino A., Tramontana P., Ta B., and Memon A. (2014), "Mobiguitar–a tool for automated model-based testing of mobile apps". 2014.

[6] Ammann P.,and Offutt J. (2008), "Introduction to Software Testing". Cambridge University Press The Edinburgh Building, Cambridge CB2 8RU, UK, Published in the United States of America by Cambridge University Press, New York, page: 10

[7] Anand S., Naik M., Harrold M. J., and Yang H. (2012), "Automated concolic testing of smart-phone apps". Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ser. FSE '12. Cary, North Carolina: ACM, 2012, pp. 59:1–59:11.

[8] Android Monkey. Retrieved on 5 January 2020, from: https://developer.android.com/studio/test/monkey

[9] Android Monkey Runner. Retrieved on 5 January 2020, from: https://developer.android.com/studio/test/monkeyrunner.

[10] Monker Recorder. Retrieved on 06 January 2020, from : https://developer.android.com/studio/test/monkeyrunner.

[11] Android Testing Framework. Retrieved on 5 January 2020, from: http://developer.android.com/guide/topics/testing/index.html.

[12] "Espreso". Retrieved on 7 January 2020, from https://developer.android.com/training/testing/espresso.

[13] Azim T. and Neamtiu I. (2013), "Targeted and depth-first exploration for systematic testing of Android apps". Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications, ser. OOPSLA '13. Indianapolis, Indiana, USA: ACM, 2013, pp. 641–660.

[14] Baride S., Dutta K. (2011), "A cloud based software testing paradigm for mobile applications", ACM SIGSOFT Software Engineering notes, Vol. 36, No.3, May 2011. DOI: 10.1145/1968587.1968601

[15] Bhuarya P., Nupur S., Chatterjee A. and Thakur R.S. (2016), "Mobile application testing: tools and challenges", International Journal of Engineering And Computer Science ISSN: 2319-7242, Volume 5 Issue 10, Oct. 2016, pp. 18679-18681. DOI: 10.18535/ijecs/v5i10.57

[16] Choi W., Necula G., and Sen K. (2013), "Guided gui testing of android apps with minimal restart and approximate learning". Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications, ser. OOPSLA '13. Indianapolis, Indiana, USA: ACM, 2013, pp. 623–640.

[17] Choudhary S. R., Gorla A. and Orso A. (2015), "Automated test input generation for : are we there yet". To appear at the 30th International Conference on Automated Software Engineering, ser. ASE '15, 2015.

[18] Dilger E. D, (March 13, 2018), The mystery of crashing apps on iOS and Android. Retrieved on 11 April 2020 from: https://appleinsider.com/articles/18/03/13/the-mystery-of-crashing-apps-on-ios-and-android

[19] De Souza, L. S. and De Aquino, G. S. (2014), "Mobile application development: how to estimate the effort?", B. Murgante et al. (Eds.): ICCSA 2014, Part V, LNCS 8583, pp. 63–72, 2014. Springer International Publishing Switzerland 2014.

[20] Elliott D, (Mar 9, 2018), "A guide to the Google Play Console" Retrieved on 5 March 2020, from:

https://medium.com/googleplaydev/a-guide-to-the-google-play-console-1bdc79ca956f.

[21] Espresso, Retrieved on 4 March 2020, from: https://google.github.io/android-testing-support-library/docs/espresso/index.html.

[22] Enge E., (April 11, 2019), Mobile vs Desktop Traffic in 2019, Retrieved from https://www.stonetemple.com/mobile-vs-desktop-usage-study/.

[23] Gao, J., Bai, X., Tsai, W. T., and Uehara, T. (2014). Mobile application testing: A tutorial. Computer. https://doi.org/10.1109/MC.2013.445

[24] Garousi, V., and Mäntylä, M. V. (2016). "When and what to automate in software testing? A multi-vocal literature review". Information and Software Technology, 76(April), 92–117. https://doi.org/10.1016/j.infsof.2016.04.015

[25] Garousi, V., and Zhi, J. (2013). "A survey of software testing practices in Canada". Journal of Systems and Software. https://doi.org/10.1016/j.jss.2012.12.051

[26] Gomez L., Neamtiu I., Azim T., and Millstein T. (2013), "Reran: Timing-and touch-sensitive record and replay for Android". Software Engineering (ICSE), 2013 35th International Conference on. IEEE, 2013, pp. 72–81.

[27] Hao S., Liu B., Nath S., Halfond W. G., and Govindan R. (2014), "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, ser. MobiSys '14. Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 204–217.

[28] Hooda, I., and Singh Chhillar, R. (2015). "Software Test Process, Testing Types and Techniques". International Journal of Computer Applications, 111(13), 10–14. https://doi.org/10.5120/19597-1433.

[29] Hu C. and Neamtiu I. (2011), "Automating GUI testing for Android applications". Proceedings of the 6th International Workshop on Automation of Software Test, ser. AST '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 77–83.

[30] Jensen C. S., Prasad M. R., and Møller A. (2013), "Automated testing with targeted event sequence generation". Proceedings of the 2013 International Symposium on Software Testing and Analysis, ser. ISSTA 2013. Lugano, Switzerland: ACM, 2013, pp. 67–77.

[31] Jorgensen P,C.. (2014). "Software Testing A Craftsman's Approach". Guest Editors Introduction, IEEE Computer (Vol. 47). DOI: 10.1109/TEST.1991.519785.

[32] Joorabchi M. E., Mesbah A. and Kruchten P. (2013), "Real Challenges in Mobile App Development," IEEE International Symposium., Baltimore MD. 2013, pp. 15-24. DOI: 10.1109/ESEM.2013.9.

[33] Kaur A. (2015), "Review of mobile applications testing with automated techniques", International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 10, October 2015. DOI: 10.17148/IJARCCE.2015.410114.

[34] Kaur A. and Kaur K. (2018), "Systematic literature review of mobile application development and testing effort estimation", J. King Saud Univ. - Comput. Inf. Sci..

[35] Kirubakaran B., Karthikeyani V. (2013), "Mobile application testing—Challenges and solution approach through automation", Proc. IEEE Int. Conf. Pattern Recognit. Inform. Mobile Eng., pp. 79-84.

[36] Kochhar, P. S., Thung, F., Nagappan, N., Zimmermann, T., and Lo, D. (2015). "Understanding the test automation culture of app developers". IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings, April 13–17. https://doi.org/10.1109/ICST.2015.7102609

[37] Kong P., Li L., Gao, J., Liu, K., Bissyande T. F., and Klein J. (n.d.), "Automated testing of android apps: A systematic literature review".

[38] Lee G., (November 23, 2014), "iOS > Android: View Life Cycle, Retrieved from http://gregliest.github.io/mobile/view-controller-lifecycle/"

[39] Li, Y. F., Das, P. K., and Dowe, D. L. (2014). Two decades of Web application testing - A survey of recent advances. Information Systems, 43, 20–54. DOI: 10.1016/j.is.2014.02.001.

[40] 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, ser. ESEC/FSE 2013. Saint Petersburg, Russia: ACM, 2013, pp. 224–234.

[41] Mahmood R., Mirzaei N., and Malek S. (2014), "Evodroid: Segmented evolutionary testing of Android apps". Proceedings of the 2014 ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE '14. Hong Kong, China: ACM, November 2014.

[42] Mehlitz P., Tkachuk O., and Ujma M. (2011), "Jpf-awt: Model checking GUI applications". Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 584–587.

[43] Memon A., Banerjee I., and Nagarajan A. (2003), "Gui ripping: reverse engineering of graphical user interfaces for testing". Proceedings of the 10th Working Conference on Reverse Engineering, ser. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 260–.

[44] Memon A. M., Pollack M. E., and Soa M. L. (2000), "Automated test oracles for GUIs". Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-first Century Applications, ser. SIGSOFT '00/FSE-8. San Diego, California, USA: ACM, 2000, pp. 30–39.

[45] Méndez-Porras A., Quesada-López C. and Jenkins M. (2015), "Automated testing of mobile applications: A systematic map and review", Conference Paper, April 2015.

[46] Monkeyrunner. Retrieved on 11 March 2019, from: http://developer.android.com/tools/help/monkeyrunnerconcepts.html.

[47] Mohammed Z., Shamlan A., HazeerA., Rizny A. (n.d.), "Challenges in Mobile Application Testing in the context of Sri Lanka".

[48] Most Popular Operating System. Retrieved on 17 March 2019, from: http://gs.statcounter.com/os-market-share.

[49] Muccini, H., Francesco, A., and Esposito, P. (2012), "Software testing of mobile applications: Challenges and future research directions", Proc. 7th Int. Workshop Autom. Softw. Test, AST 2012, Zurich, Switzerland), IEEE, pp. 29-35.

[50] Nguyen B. N., Robbins B., Banerjee I., and Memon A. (2014), "Guitar: an innovative tool for automated testing of GUI-driven software". Automated Software Engineering, vol. 21, no. 1, pp. 65–105, 2014.

[51] Orso, A., and Rothermel, G. (2014). Software testing: a research travelogue (2000–2014), 117–132. https://doi.org/10.1145/2593882.2593885

[52] Robolectric. Retrieved on 4 March 2019, from: http://pivotal.github.com/robolectric/.

[53] Robotium. Retrieved on 15 March 2019, from: http://code.google.com/p/robotium/.

[54] Sasnauskas R. and Regehr J. (2014), "Intent fuzzer: crafting intents of death". Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA). ACM, 2014, pp. 1–5.

[55] Segue Technologies (April 15, 2015), Why is Mobile Application Testing Important? Retrieved on 5 March 2019, from: https://www.seguetech.com/why-mobile-application-testing-important/.

[56] StackOverflow (2015), Retrieved on 8 March 2019, from: https://stackoverflow.com/questions/28969032/what-the-equivalent-of-activity-life-cycle-in-ios.

[57] Thu E. E., Aung T. N., (August 2015), "Developing mobile application framework by using RESTFuL web service with JSON parser", Genetic and Evolutionary Computing: Proceedings of the Ninth International Conference on Genetic and Evolutionary Computing, August 26-28, 2015, Yangon, Myanmar - Volume II.

[58] Ui Automator. Retrieved on 14 March 2020, from: http://developer.android.com/tools/testing-support-library/index.html.

[59] White L. and Almezen H. (2000), "Generating test cases for GUI responsibilities using complete interaction sequences". Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on, 2000, pp. 110–121.

[60] Yang W., Prasad M. R., and Xie T., "A grey-box approach for automated gui-model generation of mobile applications". Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering, ser. FASE'13. Rome, Italy:Springer-Verlag, 2013, pp. 250–265.

[61] Zaeem, R. N., Prasad, M. R.,and Khurshid, S. (2014). Automated generation of oracles for testing user-interaction features of mobile apps. Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014, 183–192. https://doi.org/10.1109/ICST.2014.31

[62] Zhang, D. and Adipat, B. (2005), "Challenges, methodologies, and issues in the usability testing of mobile applications", INTERNATIONAL JOURNAL OF HUMAN–COMPUTER

INTERACTION, 18(3), 293–308 Copyright © 2005, Lawrence Erlbaum Associates, Inc. DOI: 10.1207/s15327590ijhc1803_3.

[63] Franke, D., Elsemann, C., Kowalewski S., and Weise, C. (2011). "Reverse Engineering of Mobile Application Lifecycles". 18th Working Conference on Reverse Engineering.

[64] 63. L. Malisa, K. Kostiainen, M. Och, and S. Capkun, "Mobile application impersonation detection using dynamic user interface extraction," in Proceedings of the Eur. Symp. Res. Comput. Secur., 2016, pp. 217–237.

[65] 64. J. C. J. Keng, L. Jiang, T. K. Wee, and R. K. Balan, "Graph-aided directed

[66] Testing of Android applications for checking runtime privacy behaviours". Proceedings of the IEEE 11th Int. Workshop Automat. Softw. Test, 2016, pp. 57–63.

[67] L. Clapp, O. Bastani, S. Anand, and A. Aiken, "Minimizing GUI event traces". Proceedings of the ACM SIGSOFT Int. Symp. Found. Softw. Eng., 2016, pp. 422–434.

[68] Y.-M. Baek and D.-H. Bae, "Automated model-based Android GUI testing using multi-level GUI comparison criteria". Proceeding of the Int. Conf. Automated Softw. Eng., 2016, pp. 238–249

[69] H. Zhang, H. Wu, and A. Rountev, "Automated test generation for detection of leaks in Android applications". in Proceeding of IEEE 11th Int. Workshop on Automat. Softw. Test, 2016, pp. 64–70.

[70] Z. Qin, Y. Tang, E. Novak, and Q. Li, "MobiPlay: A remote execution based record-and-replay tool for mobile applications". Proceeding of IEEE/ACM 38th Int. Conf. Softw. Eng., 2016, pp. 571–582

[71] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome and D. Poshyvanyk, "Automatically Discovering, Reporting and Reproducing Android Application Crashes," 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), Chicago, IL, 2016, pp. 33-44, doi: 10.1109/ICST.2016.34.