A Literature Review on Software Testing Techniques for Smartphone Applications

Muhammad Naeem Ahmed Khan
Independent Researcher
Islamabad, Pakistan
mnak2010@gmail.com

Aamir Mehmood Mirza
Balochistan University of Information Technology, Engineering and Management Sciences
Quetta, Pakistan
mamehmood@msn.com

Raja Asif Wagan
Balochistan University of Information Technology, Engineering and Management Sciences
Quetta, Pakistan
raja.asif@buitms.edu.pk

Mahwish Shahid
University of Management and Technology
Lahore, Pakistan
mahwish.shahid@umt.edu.pk

Imran Saleem
University of Management and Technology
Lahore, Pakistan
imran.saleem@umt.edu.pk

Abstract-**Smartphone applications are getting popular and have become a necessity. There numerous smartphone applications ranging from entertaining to gaming and from utility to mission-critical. Almost everything on the web is now in hands of Smartphone users, which makes this domain very important and its quality should not be compromised. Achieving the desired quality is not an easy task for the mobile platform as it has its limitations. To produce a quality app, developers and testers need to test and assess the app in numerous ways to ensure the best trait of the application. In this concern, some efficient and mature techniques are required to test smartphone applications. In this study, the techniques, approaches, and models to assess mobile apps covering major prospects and angels to test mobile apps are identified. Our focus is on assessing the existing techniques and to evaluate them on standard validation parameters.**

Keywords-Android; model based testing; functional testing; app testing; functional refactoring

I. INTRODUCTION

Smartphone applications are getting popular and have become a necessity. From banking to healthcare or from gaming and utility to mission-critical, there is a huge pool of smartphone applications [1, 2]. Achieving the desired quality is not an easy task for the mobile platform as it has its limitations such as processor, battery etc. To produce a quality app, developers and testers need to validate it in numerous ways to ensure the best trait of the application [3]. In this concern, some efficient and mature techniques are required to test mobile applications. Mobile applications have their quirks and challenges regarding testing, such as the high number of different events that need to be tested [4]. Security is also an important aspect of smartphone applications [5]. These challenges mostly rely on the mobile platform, but some challenges arise due to the interoperability of the mobile platform to other platforms like the web, third party systems, and the cloud. In this study, the techniques, approaches, and models to assess mobile apps covering major prospects and angels to test mobile apps are identified. Our aim is to assess the existing techniques and to evaluate them on standard validation parameters.

II. RESEARCH METHODOLOGY

This review was conducted according to the guidelines proposed by [6]. For this purpose, we have formulated a search string and executed this search string on IEEE Explore, ACM and Science Direct to identify research studies published from 2007 to 2020. By reviewing the title and the abstract we have initially selected 98 research papers for full text reading. From this initial database of research studies, we selected 19 research papers to include in this study which were aimed at testing smartphone applications. The database of the selected research papers covers journal and conference papers about testing techniques for smartphone applications. Most of the research studies are primarily focused on theoretical reports, case studies, field studies, and experience reports.

III. LITERATURE REVIEW

A. Security and Malware Testing

A study presented APSET for detecting the intent-based vulnerabilities of Android applications. APSET takes vulnerable patterns proposed by domain experts and system specifications [5]. The major contribution of this study is test case generation via the automatic generation of partial specifications from applications. APSET also detects issues in the data on the basis on intent mechanism. APSET was tested on over 70 Android applications and it detected 62 vulnerabilities which could be exploited by hackers or attackers to crash the application. This tool is founded over the model-based testing technique along with the support of the ioSTS model to generate patterns besides the reverse engineering of

Corresponding author: Mahwish Shahid

system class diagram and specification to generate test cases for the application under test. The limitation of the proposed tool is that it only works over intent-based loopholes and pitfalls. APSET cannot be used for any other vulnerability. A novel hybrid technique was proposed to detect malware in Android applications based on static and dynamic analysis [7]. The proposed technique requires efficient data processing (pattern generation and detection). System calls are used to generate patterns for malware and normal apps which are done through the support of Android OS manipulation. The advantage of this approach is that it does not spend much time as it takes in the static analysis, neither it consumes a lot of resources as does the dynamic analysis. Some disadvantages include that it needs to constantly gather new malware and benign apps to keep detection accuracy because new types of malware continue to emerge. This approach is based on the difference between malware and benign apps in runtime system calls. More normal and malware patterns are always needed. Second, due to limited computing and storage resources, it is not suitable to perform large-scale data processing in the mobile phone. To achieve real-time detection, this approach requires the mobile phone to have powerful computing capabilities and sufficient data storage. Authors in [8] presented a test automation technique for mobile platforms based on observation, extraction, and abstraction of the running SUT by using its GUI widget. The abstraction used in this tool is used to create a scalable state machine model using event-based test coverage criteria, which automatically creates test cases for the SUT. The study demonstrates that the generated test cases were effective and useful in detecting serious issues and defects in the applications. Using the mentioned approaches, the authors developed a fully automatic tool for the detection of bugs and errors in Android applications. This study also compares two famous Android application testing tools: Monkey and Dynodroid. They configured Monkey and Dynodroid to test the same four mobile applications which they tested with Mobiguitar under the same parameters and inputs. An experiment demonstrated that Monkey and Dynodroid both did not find all of the exceptions which their proposed tool did.

B. Cloud-based Testing

A systematically review the state of the knowledge of the empirical studies is presented in [9]. The study focuses on mapping the testing techniques for mobile applications. Additionally, the study emphasizes the need for testing metrics to be included and adhere to address mobile application testing lifecycle conformance. The major lags in the mentioned techniques for a smartphone application testing lie in the automation of testing. According to the authors, this is an emerging and future of mobile and other testing, but very few of them implemented this technique over complex applications. Automated testing techniques perform well over small to medium and simple mobile applications, but very little work is done over the implementation and analysis of this technique over complex mobile applications which put a question mark over its credibility and reliability.

C. Test Automation

Authors in [10] proposed a novel framework for comparing automatic testing techniques for smartphone applications. The salient characteristics of every technique were picked to develop the parameters. A comparison was done among online testing techniques and in result, a general framework was proposed based on the Unified Online Testing Algorithm. The authors found that the random technique is more effective than an active learning technique, but it is approximately 100 times more expensive due to the nature of the experimental setup. Active learning was cheaper and more efficient than the random one in some cases, but in most of the cases the random testing technique was better in execution but more expensive in cost. Authors in [11] presented an experimental study to analyze and evaluate the MBT approach in modeling, concretization, and execution of automated tests in mobile applications. Along with the usage of MBT they adopted the Event Sequence Graph (ESG) to design their test model. For the implementation of the test cases, they used the Robotium framework. This study evaluated the perks and shortcomings of using MBT as an approach to automate test execution for this particular platform. Several perks and challenges were identified including: automatic generation of test cases, capacity to detect faults, improvement in test quality, test time and cost reduction, and evolution of test models. The challenges in the usage of MBT in the mobile application that are related to this domain are: difficulties in test modeling and particularities in the concretization of test cases in mobility context and in-depth knowledge of Robotium.

The design and implementation of the mobile TaaS system called MTaaS is presented in [12]. MTaaS is an infrastructure for mobile application testing on the cloud which provides large scale remote mobile application testing for the Android platform. Some issues regarding resource allocation and sharing were discussed as this study focuses on the cloud to implement the proposed system where resource allocation and management is a very important factor to be handled. This issue is resolved through a hybrid model that aims to improve the system performance while reducing cost. The system was tested on different real time scenarios and use cases. MTaaS was compared to Perfecto Mobile, TestDroid, YiceYun, Testin, and UTest under the same parameters and inputs. The results showed that MTaaS performed satisfactorily. Further, some limitations were discussed like the security and privacy of the user's data, security threats to multiple virtual machines, and intrusion detection. Another issue that was highlighted was the lack of standards in mobile test environments and test automation for mobile application testing with the addition to the lack of well-defined test models and coverage criteria for cloud-based application testing. Authors in [13] presented a tool named SIT to test self-adaptive applications. SIT framework stands on Abstract Trace (AT) and Trace Segments (TS). Test case generation was achieved by sampling-based test generation. As per experimental results, the SIT improved defect detection significantly. SIT was evaluated and analyzed over three online available context-sensitive and self-adaptive applications: Robot car, Phone Adaptor and SECONDO. The study implemented Random Testing (RT) and Dynamic Symbolic Execution (DSE) for comparison purposes. Overall the SIT improved detection rate by 22.4-42.2%. Despite all these advantages and good results, the SIT has some limitations. It currently relies on application specific support,

which means if an application is running in a particular environment, that environment should not be easily changed or manipulated during testing. The improvement room is also available in the proposed approach in a way that assertions that were used to define program failures were composed manually in the application. It might be possible to derive these assertions automatically from specifications.

In [14], a GUI based automated testing tool called SPAG-C is proposed which uses a record-replay technique to perform GUI testing. SPAG-C used event-based and smart wait function to eliminate the uncertainty of the reply process and by using GUI layout information to verify the results testing process produces. The experimentation showed that SPAG-C maintains the accuracy to 99.5% in addition to the reduction of the time required to record test cases by verifying the process automatically with the increased reusability of the test oracles without compromising the accuracy. The proposed tool was compared with Histogram, SURF, and Template matching tools and their techniques and results proved that SPAG-C outperformed them. The novelty of the SPAG-C is that for some apps only a small portion of the screen is changed when the app responds to an event. For such apps the tester can select the region of interest in the screen of the device and then SPAG-C will verify it which saves time to fully verify the screens once again. As a limitation, there is the problem that applications with non-deterministic GUI cannot be tested by SPAG-C. This is because the camera cannot extract the necessary elements from the screen for later comparison and matching as AUT screen keeps changing in video playing applications or gaming apps. This approach works on the image capturing mechanism so the camera which is used to capture screenshots of all UI could be affected by external factors of the environment like light, exposure etc. A controlled environment is required to execute this procedure. Authors in [15] presented an empirical study in which MBT was applied to mobile applications to examine the effectiveness of their approach over mobile systems and applications. The study uses EFSM to model a mobile app and implemented a command-line tool "Kelevra" along with Appium, an automation tool. Gestures, clicks and keyboard inputs are examples of methods that can then be applied to the retrieved UI element objects. The MBT approach was also previously tested over an app "GMSEC", which is quite simple as compared to "Quiz-Up". The experimental setup exhibits that MBT pays well off over the effort it requires. Experiments show that applying MBT found non-trivial bugs and defects in Quiz-Up which was already being tested, proving the effectiveness of this approach towards mobile systems or applications. A possible extension to this would be to minimize the manual steps even more. Another option would be to implement a language to describe a SUT and its possible outcomes. For mobile applications, we can describe the UI elements and patterns in a particular view or scene under the test. The constructed textual description could then be translated into a model representation such as the EFSM. The key limitation observed is that the proposed technique is suitable for testing small-sized apps.

Authors in [16] presented an adaptation model for testing mobile applications which is comprised of two sub-parts, Mobile analyzer and Test Mobile [16]. The study was inspired by a framework used by web app testing, Reweb and Testweb, a tool for analysis, testing and restructuring application. Refactoring is a key step in this proposed technique as refactoring minimizes code size without affecting the functionality of an app. The major focus of this study was to minimize the test effort by minimizing the test cases of an app, which can be achieved through refactoring AUT (Application Under Test), so that the transitions and paths of applications can be minimized which ultimately results in less test cases and efforts to test an app without compromising the coverage. Based on this model, test cases were generated. At this point, applications entered the second module, Test Mobile, and generated test cases were selected upon defined criteria. After test case selection, the expected outcome and run time output were compared after the execution, and the result decided whether the test passed or failed. Reweb and Testweb were initially used for web domain applications and they were designed for a particular domain so they performed well. The advantage of this technique is that it minimizes test case space without compromising coverage. Further advancement in this technique can be made by automating the process of refactoring which eases its use and lessens human involvement and effort on refactoring. Authors in [17] presented an MBT approach for test case generation for smartphone applications. State machines were used for modeling and test case generation. Test cases were generated through SPIN, an automatic tool for test case generation through model usage. The proposed approach used XML based transformations to translate the test cases to some executable form to activate the applications under test. For experimentation, Facebook and YouTube applications were used. The proposed approach was tested with other model-based approaches and techniques which include APSET, MobiGUITAR, and SwiftHand. This approach uses the view state machine model to model any application, which involves excessive mathematics and prior working knowledge of state machines, as these models become very complex in large size programs. The second issue is that there is a lot of involvement of third-party tools which are assisting in the basic three steps of this approach.

Authors in [18] proposed a grey box approach for automatically reverse engineering GUI-models of mobile applications. Their system, Orbit, uses a finite state machine for model generation and the results of an empirical evaluation on several real apps were presented. At first, the authors used static analysis of the application's source code to extract the set of user actions supported by each widget in the app GUI. Then, a dynamic crawler was used to reverse engineer a model of the app, by systematically exercising the extracted actions on the live app. Orbit was evaluated on 3 parameters, test coverage, time consumption, and precision. It was compared with Monkey, MobiGUITAR, and Android GUI Ripper in experimental steps. A limitation in this approach is that it requires a lot of manual work, consisting in manually selecting attributes of the executable components to compose the visual observable states for the GUI.

Authors in [19] proposed a technique to test mobile applications using reverse engineering and pattern recognition of mobile AUT. The process was based on the automatic and dynamic exploration of the apps' GUI. To support the dynamic

exploration, static analysis was also conducted to verify how they are executed. GUI widgets were examined through their execution and calls. The achieved patterns decided the errors and bugs, if found. The major contribution of this work is a reverse engineering approach to identify occurrences of behavioral patterns in mobile applications and a dynamic, run time, and on the go testing approach based on the application of test patterns associated with behavioral patterns identified in the mobile application. The proposed approach was found to perform better than the MBT approach as most of the GUI MBT approaches use reverse engineering for model gain. A drawback of this study is that the pattern behavior judgment is not easy and can be a tedious task if done manually. Since this study reverse engineers the app under test and analyzes its patterns statically, this static task of pattern recognition and extraction is not easy as apps get more complex day by day. Future work in this study could be the automation of pattern identification and recognition, which would reduce and minimize execution time and minimize human error.

Authors in [20] proposed test adapters to test GUIs through automated testing of industrial applications. Test generations through test adapters can be applied at the unit, integration and system test level. Authors in [21] proposed the A3E approach to use static, taint style, data flow analysis on the app to develop a high level control flow graph that captures legal transitions on the app's screen. The experiment on 25 popular apps achieved 59.39-64.11% activity coverage and 29.53-36.46% method coverage. Model-based software testing can be used to automatically produce test cases from a formal model describing the SUT. In addition to conventional test automation, it may increase the quality of testing and reduce the resources needed. A case study was presented to illustrate the ability of a MBT to produce long-term test cases and run parallel tests on multiple smartphone devices in [22]. Authors in [23] introduced a novel method designed to identify irregular network traffic activities in a multimedia app and how different user experiences will lead to unexpected traffic patterns. It makes the generation of a test suite composed of a huge number of test cases that can be executed and measured with adequate automation. Instead of random interactions, this test suite represents realistic user behaviors, which may reveal unforeseen consequences to users.

IV. CRITICAL EVALUATION

A systematic literature review was performed in order to find, analyze, and classify papers which focused on testing mobile applications. The aim of the current study was to provide practitioners and researchers a clear view of the state of the art so that they can easily find existing solutions pertinent to their issues. The papers have been classified based on the focused area, testing techniques/models, tool(s) (used and proposed), platform/OS, tested applications domain, and validation methods and parameters. A summary of the approaches and techniques is presented in Table I.

TABLE I. CRITICAL EVALUATION

Ref.	Focused Area	Testing techniques / models	Tools	Platform / OS	Tested applications domain	Validation parameters	Comparison with other techniques, tools, and models
[4]	Test oracle automation using model driven approach	-State machine -Object management group -Class diagram -Sequence diagram	xUnit (used)	Android	Utility app	-Bug detection -Test case execution time -Model transformation time	-
[5]	Security testing for Android based smartphones	-Reverse Engineering of app class diagram -ioSTS model	APSET	Android	-General utility -Location based -Entertainment -Dictionary	-Vulnerability path coverage (80%) -Bug detection (88%) -Test terdicts (23%)	Tools: Notepad, Google Map, YouTube and Searchable Dictionary
[7]	Malware detection in Android smartphones	-Static analysis -Dynamic analysis	None	Android	-Learning apps -Utility tools -Games	-Malware detection -Accuracy up to 90%	Tools: AndroGuard and DroidMat
[8]	Model based automatic test cases generation and execution for mobile platforms	-Model based testing -Reverse engineering -FSM	MobiGUITAR	Android	-Android application testing tools	-Exception handling -Defect detection -Code coverage (70%)	Tools: Aardict, Monkey and Dynodroid Techniques: Model based testing, random testing
[9]	Mapping mobile application testing techniques to self-defined classification schemes.	-Model based -Data driven -Search based -Reverse engineering -Contextual fuzzing -GUI based testing -Automated testing -Scripted UI testing -Event based testing	None	-Android -Symbian -Windows -Silverlight	-News reader -Spreadsheet applications -Advertising -Mobile learning -Sales force automation -Android testing	-Usability testing -Security testing -Test automation -Context awareness -General mobile	Tools: AppDoctor, JPF Android, MobileTest, TestDroid, DroidChecker, Caiipa, Mobiguitar, and AppInsight Techniques: Model based, GUI based, automated and event based testing
[10]	Comparing and evaluating automatic testing techniques for Android mobile apps.	-Event based testing -Graph based search	None	Android	-Mobile games -Calculator -Shopping list manager -Recipe manager -Task managing -Note taking -Battery viewer	-Avg. code coverage (0.035%) -Avg. testing cost (7.86ms)	Tools: Aarddict, SimplyDo, TicTacToe, Trolly, TomDroid, TaskMan, BitesandManPages Techniques: Active learning and Random Testing. Model: GUI Tree

[11]	Test automation for mobile applications	-Model based testing -Event Sequence Graph	Robotium	Android	Contacts and address manager	-Avg. event sequence execution time (99.93s) -Fault detection rate -Cyclomatic complexity (1,22)	Tools: Address Book App
[12]	Cloud based testing for Smartphone applications	-Mobility testing -Security testing -Functional testing -GUI testing -Service oriented testing	MTaaS	-Android -Cloud	Android application testing tools	-Performance testing -Avg. response rime: 181ms -Avg. request hit rate: 35.6/s -Avg. error ratio: 2.793% -Test Case success rate (100%)	Tools: Perfecto Mobile, TestDroid, YiceYun, TestinandUTest Infrastructure approach: Crowd sourcing, emulation based, and device based
[13]	Testing of self-adaptive mobile applications	Sampling based interactive testing	SIT	Android	Context aware applications	-Number of bugs: 91.4% -Bug detection rate: 19.8% -Test time efficiency: 82.6% -Branch coverage: 12.3-47.9% -Test effectiveness: 95.2%	Tools: Robot Car, SECONDO, Caiipa and Phone Adapter Techniques: Random testing and DSE
[14]	Assessing accuracy, efficiency and reusability of testing oracles for Android devices.	-GUI testing -Event batching -Smart wait function -Record-replay	SPAG-C	Android	Android application testing tools	-Accuracy (99.5) -Efficiency (50-75%) -Reusability (approx. 5h) Based on -False positive rate -False negative rate	Tools: SPAG and MonkeyRunner Techniques: Histogram, SURF and Template matching
[15]	Mobile application testing through model-based testing	EFSM model	Used: Appium Proposed: Kelevra	Android	Learning Apps	-Fault detection rate	Tools: GMSEC
[16]	An adaptive model for mobile application testing	Source code refactoring	ReWeb&Test Web (used)	Android	Calculator	-Bug detection -Test suite/code minimization	Tool: Calculator
[17]	A model-based testing approach for generating test cases for Android apps.	State machine	SPIN (used)	Android	N/A	-App coverage -Testing time duration	Tools: MobiGUITAR& APSET Techniques: SwiftHand
[18]	A model-based method of automatically reverse engineering GUI-models of mobile applications	Finite state machine	ORBIT	Android	-Business -Productivity -Entertainment -Literary	-Test coverage: 79% -Testing time duration: 51% -Precision	Tools: Monkey, Android Guitar & Android GUI Ripper Technique: Depth First Search & Forward Crawling
[19]	An automated GUI testing approach to find defects/bugs in mobile applications	-GUI dynamic analysis -Static analysis	Dalvik VM (used)	Android	-E health -Mobile games -Business -Communication	Defect rate	Tool: Monkey Techniques: Model based testing
[20]	Applied code level testing to test rich GUI based applications.	-Unit testing -GUI testing -Manual testing	Randoop (used)	Windows	Industrial projects (case studies)	-Defect detection rate (51%) -Test case generation (153) -Coverage (63%)	Tools: jUnit Techniques: Manual testing, jUnit Automated testing (Java FX)
[21]	Systematic exploration of Android apps for testing purpose.	-Black box testing -DFS -Targeted exploration -Dynamic analysis	A3E (proposed)	Android	-Entertainment -E commerce -Media -Social -Music -Health	-Method coverage -Activity coverage -Exploration time	Techniques: Manual testing
[22]	Model based GUI testing approach for testing Smartphones apps	-MBT -GUI testing	TEMA Test Engine (used)	Android	-Camera -Messenger	-Bug detection -App state coverage	-
[23]	Identification of abnormalities in user interaction with multimedia apps using model based approach.	-MBT -State machine diagram	MVE (proposed)	Android	Music	-No. of executed test cases -State coverage	-

V. CONCLUSION

In this study, several techniques and approaches in addition to models and tools were reviewed for smartphone app testing. We found a pool of testing strategies to assess the quality of mobile apps of various natures. While analyzing and evaluating the reviewed approaches for mobile app testing, we found the model-based approach more convenient and promising to test mobile apps because of its appealing approach to model the overall design of a system [24]. The main advantage of using the model-based approach is that the testers are well satisfied with the recused efforts and the level of test cases it helps produce [25]. Another advantage to adopt model-based testing is that it supports several automated tools for test case generation and execution. In this study, we have presented an abstract model for testing smartphone apps which support

semi-automated testing. A potential future work in this context could be to evolve a fully automated approach in the form of a testing tool which takes an app as an input and generates test cases according to the chosen model. The proposed tool may possess more than one model for modeling that app and would present the test results after executing the generated test cases in accordance with the selected model. The proposed approach would significantly lessen the human efforts required to test mobile apps and would also minimize the chances of human error.

REFERENCES

[1] Z. U. Rehman and F. A. Shaikh, "Critical Factors Influencing the Behavioral Intention of Consumers towards Mobile Banking in Malaysia," *Engineering, Technology & Applied Science Research*, vol. 10, no. 1, pp. 5265–5269, Feb. 2020, https://doi.org/10.48084/etasr.3320.

[2] M. C. Lam, M. Ayob, J. Y. Lee, N. Abdullah, F. A. Hamzah, and S. S. M. Zahir, "Mobile-based Hospital Bed Management with Near Field Communication Technology :," *Engineering, Technology & Applied Science Research*, vol. 10, no. 3, pp. 5706–5712, Jun. 2020, https://doi.org/10.48084/etasr.3527.

[3] M. Linares-Vásquez, K. Moran, and D. Poshyvanyk, "Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 399–410, https://doi.org/10.1109/ICSME.2017.27.

[4] B. P. Lamancha, M. Polo, D. Caivano, M. Piattini, and G. Visaggio, "Automated generation of test oracles using a model-driven approach," *Information and Software Technology*, vol. 55, no. 2, pp. 301–319, Feb. 2013, https://doi.org/10.1016/j.infsof.2012.08.009.

[5] S. Salva and S. R. Zafimiharisoa, "APSET, an Android aPplication SEcurity Testing tool for detecting intent-based vulnerabilities," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 2, pp. 201–221, Apr. 2015, https://doi.org/10.1007/s10009-014-0303-8.

[6] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University, EBSE 2007-001, 2007. Accessed: Dec. 09, 2020. [Online]. Available: http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf.

[7] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in Android," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 22–31, May 2017, https://doi.org/10.1016/j.jpdc.2016.10.012.

[8] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "MobiGUITAR: Automated Model-Based Testing of Mobile Apps," *IEEE Software*, vol. 32, no. 5, pp. 53–59, Sep. 2015, https://doi.org/10.1109/MS.2014.55.

[9] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," *Journal of Systems and Software*, vol. 117, pp. 334–356, Jul. 2016, https://doi.org/10.1016/j.jss.2016.03.065.

[10] D. Amalfitano, N. Amatucci, A. M. Memon, P. Tramontana, and A. R. Fasolino, "A general framework for comparing automatic testing techniques of Android mobile apps," *Journal of Systems and Software*, vol. 125, pp. 322–343, Mar. 2017, https://doi.org/10.1016/j.jss.2016.12.017.

[11] G. de Cleva Farto and A. T. Endo, "Evaluating the Model-Based Testing Approach in the Context of Mobile Applications," *Electronic Notes in Theoretical Computer Science*, vol. 314, pp. 3–21, Jun. 2015, https://doi.org/10.1016/j.entcs.2015.05.002.

[12] C. Tao and J. Gao, "On building a cloud-based mobile testing infrastructure service system," *Journal of Systems and Software*, vol. 124, pp. 39–55, Feb. 2017, https://doi.org/10.1016/j.jss.2016.11.016.

[13] Y. Qin, C. Xu, P. Yu, and J. Lu, "SIT: Sampling-based interactive testing for self-adaptive apps," *Journal of Systems and Software*, vol. 120, pp. 70–88, Oct. 2016, https://doi.org/10.1016/j.jss.2016.07.002.

[14] Y. Lin, J. F. Rojas, E. T.- Chu, and Y. Lai, "On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 957–970, Oct. 2014, https://doi.org/10.1109/TSE.2014.2331982.

[15] V. Gudmundsson, M. Lindvall, L. Aceto, J. Bergthorsson, and D. Ganesan, "Model-based Testing of Mobile Systems -- An Empirical Study on QuizUp Android App," *Electronic Proceedings in Theoretical Computer Science*, vol. 208, pp. 16–30, May 2016, https://doi.org/10.4204/EPTCS.208.2.

[16] M. Ahmed, R. Ibrahim, and N. Ibrahim, "An Adaptation Model for Android Application Testing with Refactoring," *International Journal of Software Engineering and Its Applications*, vol. 9, no. 10, pp. 65–74, Oct. 2015, https://doi.org/10.14257/ijseia.2015.9.10.07.

[17] A. R. Espada, M. del M. Gallardo, A. Salmerón, and P. Merino, "Using Model Checking to Generate Test Cases for Android Applications," *Electronic Proceedings in Theoretical Computer Science*, vol. 180, pp. 7–21, Apr. 2015, https://doi.org/10.4204/EPTCS.180.1.

[18] W. Yang, M. R. Prasad, and T. Xie, "A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications," in *Fundamental Approaches to Software Engineering*, Berlin, Heidelberg, 2013, pp. 250–265, https://doi.org/10.1007/978-3-642-37057-1_19.

[19] I. C. Morgado, A. C. R. Paiva, and J. P. Faria, "Automated Pattern-Based Testing of Mobile Applications," in *2014 9th International Conference on the Quality of Information and Communications Technology*, Guimaraes, Portugal, Sep. 2014, pp. 294–299, https://doi.org/10.1109/QUATIC.2014.47.

[20] R. Ramler, G. Buchgeher, and C. Klammer, "Adapting automated test generation to GUI testing of industry applications," *Information and Software Technology*, vol. 93, pp. 248–263, Jan. 2018, https://doi.org/10.1016/j.infsof.2017.07.005.

[21] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," in *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, New York, NY, USA, Oct. 2013, pp. 641–660, https://doi.org/10.1145/2509136.2509549.

[22] R. Dev, A. Jääskeläinen, and M. Katara, "Model-Based GUI Testing. Case Smartphone Camera and Messaging Development.," *Advances in Computers*, vol. 85, pp. 65–122, 2012, https://doi.org/10.1016/B978-0-12-396526-4.00002-3.

[23] A. R. Espada, M. del M. Gallardo, A. Salmerón, and P. Merino, "Performance Analysis of Spotify® for Android with Model-Based Testing," *Mobile Information Systems*, vol. 2017, Feb. 2017, Art. no. 2012696, https://doi.org/10.1155/2017/2012696.

[24] A. M. Mirza and M. N. A. Khan, "An Automated Functional Testing Framework for Context-Aware Applications," *IEEE Access*, vol. 6, pp. 46568–46583, 2018, https://doi.org/10.1109/ACCESS.2018.2865213.

[25] S. Mohacsi, M. Felderer, and A. Beer, "Estimating the Cost and Benefit of Model-Based Testing: A Decision Support Procedure for the Application of Model-Based Testing in Industry," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, Funchal, Portugal, Aug. 2015, pp. 382–389, https://doi.org/10.1109/SEAA.2015.18.