An Attempt towards Model Based Functionality, Performance, and Compatibility Testing

Najam us Saqib, Sara Shahzad
University of Peshawar
Pakistan
nsaqibstd@uop.edu.pk
sara@uop.edu.pk

ABSTRACT: *Mobile applications have proliferated internet market due to abundant development of smart mobile phones and mobile devices. Testing of these mobile applications is crucial before they are made available to the customers. Testing mobile applications has remained a big challenge for software professionals because of diverse features of mobile phones as well as limitations which come with small size and portability. In this context testing of the application's GUI is of utmost importance. GUI is tested for for efficiency, reliability and user friendliness. A number of testing models and techniques have been proposed to automate GUI testing to reduce testing time, but still there is no technique which can be considered as standard. This paper presents an analysis of available GUI testing techniques to expose their limitations and critical issues related with GUI testing automation, cross platform testing, GUI modeling, and automatic test cases generation and execution. To handle these issues TriTest, a model based approach is proposed which aims to automatically test functionality, compatibility, and performance of mobile apps.*

Keywords: Software Testing, Graphical User Interface (GUI), Application Under Test (AuT), User Interface (UI) Model Based Testing (MBT), Test Cases

DOI: 10.6025/stj/2018/7/2/52-62

Received: 21 April 2018, Revised 27 May 2018, Accepted 6 June 2018

1. Introduction

Software testing is considered to be one of the most critical phases of software development. In testing, an application is examined to maintain correctness by confirming functionality of application against user specifications, and errors are removed. As the main goal of any software application is user acceptance which is only possible if that software application is correct and complete for its functionality. Applying different Software testing techniques is the only way to achieve high quality and user acceptable application.

The growth of mobile devices is increasing with a high rate. According to Gartner [1], worldwide sales of smart phones in 2015 totaled 330 million sets, an over 10 percent increase as compared to sales in 2014, as the usage of smart phones has exponentially increased compared to feature phones. Android is the most popular operating system in the smart phones market. In first quarter of 2017 and has the largest market share of 64.2 percent [2]. Similarly development of mobile software applications is also accelerated [3] but producing high quality mobile phone applications is still a challange. Anderia et al. [4] says testing mobile apps is a big challenge for software professionals because of limited computational resources and factors including mobile phones hardware and software configuration, touch screen, diverse network protocols, operating system, user interface specifications, data transmission modes and energy consumption of the mobile device, etc. Furthermore, the usage of mobile applications is directly proportional to the quality of application. Princeton Survey Research International conducted a survey [5] which concludes that over 75 percent of available mobile applications (apps) go unused. If the application is not up to the quality of users and not fulfilling their needs the users will reject this application and try another one as they have a lot of choices available. This perceived quality of the application is dependent on the compatibility of the application. For apps to make their business and place in raising competitor app stores proper app testing is necessary.

Many approaches have been proposed for automatic testing of mobile apps. Among all of these approaches cloud based, model based and testing through GUI analyzing (discussed in Section 4) are considered to be the main approaches to achieve automatizing in testing. Among these three, model based testing approach is selected as optimal for complete functionality testing and automatically test cases generation and execution (as discussed in section 5). Automatic test cases generation and execution can save time and cost but have nothing to do with the acceptance of application among users. Key to application acceptance among users is that applications have correct functionality, compatibility, and are performance efficient. Model based testing techniques provide complete functionality testing but do not test compatibility and performance of an application.

TriTest is an MBT approach to automatically test functionality, performance and compatibility of an application. It is based on SlumDroid [6]. In TriTest first navigation model is created using GUI ripping technique. To test functionality, test cases are automatically generated from the created model and executed on the basis of information provided by GUI analyzer about the GUI components such as input fields. Performance of an application will be tested by calculating response time, throughput, and resources (memory and processor) consumed by an application when test cases starts execution. Installability, adapt-ability, and replace-ability of application over that devices is measured to evaluate compatibility of an application. These three are considered as characteristics of compatibility in ISO/IEC 9126-1 [7].

The paper is structured as follows. Section 2 explains different methodologies, tools and techniques used for Mobile GUI testing and different problems in these methodologies are discussed. In Section 3, different issues in existing GUI testing techniques and tools are presented. In Section 4 three main open areas and limitations in different approaches regarding automated mobile apps testing are discussed. In Section 5 evidence are provided for the selection of best approach for GUI testing of mobile apps. In Section 6 proposed approach for achieving model based functionality, performance, and compatibility testing is discussed. Section 7 presents the conclusion.

2. Mobile Gui Testing Techniques and Tools

GUI Testing of mobile apps is hot area of research because of its importance and complexity. Many techniques and tools are presented for testing various aspects of GUI of mobile apps. Following sections present a literature review on the different tools techniques available for testing GUI of mobile apps.

1) Mobile Application Testing (MAT) Tool: This is cloud based framework for mobile application testing. It performs different types of tests on mobile apps [8] [9] [10]. C. Mano et al [11] presented a framework in which MAT tool is used to test the performance and the compatibility of mobile applications on different mobile devices.

2) MobiTest: MobiTest [12] is a cross-platform testing tool for mobile applications. MobiTest compares UI of mobile applications of on different platforms and then create a single suite of test cases for all platforms. These test cases are then executed on all platforms simultaneously and results are recorded.

3) Parallel Android Testing System (PATS): Hsiang-Lin Wen et al. [13] introduce parallel android testing system (PATS) to dynamically test and analyze GUI of an app under the cooperation of master and slave nodes.

4) ReWeb and TestWeb: Maryam et al [14] presented ReWeb and TestWeb techniques for analyzing android apps and automatic generation and execution of test cases.

5) Fuzz Testing: In Fuzz Testing [15], the prior knowledge about the application is not required. It gives unexpected and random events to the application and records the behavior of application.

6) Record and Replay (*R* and *R*): In *R* and *R* [15] while testing on an application, actions (keystrokes and gestures) are recorded. The recorded actions can be replayed automatically when the test is performed next time.

7) T+: In T+ [15] different events related to an app generated from app testers and developers are recorded and analyzed to generate test cases. Then these test cases are validated to select actionable test cases.

8) AndroidRipper: AndroidRipper [16] is proposed to automate the testing process of android applications. Android ripper traverses the UI of the application automatically and then dynamically analyzes it. Test cases generated on the basis of automatic analysis process of android ripper and events found during the ripping process. These test cases are the executed.

9) SlumDroid: SlumDroid [6] is the modified version of AndroidRipper [16]. SlumDroid uses GUI Ripping technique and also create navigation model of application under test (AuT). It emulates its physical interactions by interacting with the UI elements of the application. Changes in the GUI screen are determines and redundant explorations is avoided. SlumDroid captures screenshots during the ripping process and also build a GUI tree. SlumDroid produce perturbed inputs based on created GUI model. Screenshots and GUI tree is analyzed and perturbed inputs are produced which are then used to test application automatically.

10) Test Cases Migration: Shauvik [17] presented Test Cases migration technique in different platform based versions of an app are given as input to migrate test cases from one mobile platform to another mobile platform. Different actions against each platform are performed and perfect match against actions for the app of both platform are recorded. Then for the matched actions (similar actions) the test cases from the source platform are migrated to target platform.

11) Combinatorial Approach: It is proposed by Sergiy Vilkomir et al [18] to test mobile applications. In combinatorial approach different mobile devices, with distinct features are picked for testing the GUI and other features of mobile applications. In this approach two methods, each-choice and pair-wise is used for selection of devices to ensure complete coverage of the features of all devices.

12) User Interface Exchange: A/B (alpha beta) testing compares different versions of application by sharing it among users and analyzes their behavior when these are being used. Florian et al. proposed a technique name as remote user interface exchange. This technique enable A/B testing for android mobile applications. In this technique, an android application is developed and shared on an app store so that users can download and use it. The user is notified about the new UI when designed. The user downloads the update without requiring an update in the app store from the app's production server. This is the way how different designs are distributed among different users [19].

13) Optical Character Recognition (OCR) Ripper: Automated test cases generation and execution for mobile apps can be made possible using software GUI modeling for mobile device using Optical Character Recognition (OCR) [20]. OCR uses image processing technique to extract character information such as clickable widgets and windows connected with these widgets. OCR ripper uses optical character recognition to travel across through the GUI of mobile application, create GUI model of that application, and creates test cases on the basis of that model.

14) Mobile Interface Modeling (MIM) Language: Model driven development (MDD) reduces coding errors, development time, and usability of an app as it is purely based on user requirements. To enable MDD in mobile apps, Sebastian et al [21] presented mobile interface modeling (MIM) language considering the good practices of software provider in usability of mobile apps. MIM has been designed to specify all the characteristics of final UI of an application. Then these specifications are communicated to the developers, diminish the time taken by the developers in the specification of GUI.

15) Experience Sampling Method (ESM): For A/B testing of mobile apps, ESM [22] is used to evaluate mobile application by

catching the behavior of users. In ESM, two UI versions of mobile applications and data is recorded from users at time when users interact with that application. The recorded information is then evaluated for A/B testing of those user interfaces.

16) Pattern Based GUI Testing: PBGT [23] [24] is a model based mobile applications testing technique. PBGT combines the techniques of pattern identification and reverse engineering to test UI Patterns and increase systemization and reusability in android mobile applications testing.

17) ADAutomation: ADAutomation [25] is a framework which consists of a tool chain for automatic generation and execution of test cases for mobile apps. In this framework user behavior model is extracted from UML activity diagrams. On the basis of extracted model automatic test cases are generated using test script generator which translates user actions into corresponding test cases. All the generated test cases can be executed using test engines and results are recorded and reported.

3. Issues in Available Gui Testing Techniques

The following subsections expose some of the critical issues concerning the GUI testing techniques and tools presented in the previous section.

3.1 Platform Testing

Testing mobile apps against different platforms and different versions of operating systems is very challenging, expensive, and time consuming activity. Researchers propose different techniques to test mobile apps on cloud framework to handle this diversity. These techniques have some limitations. MAT tool presented by [11] tests only android applications. The tool also does not perform load and interruption testing. It only measure compatibility and performance of application over different devices but GUI issues and errors are not reported. MobiTest [12] does not solve all the problems associated with the automated testing of mobile applications because it only generate test cases for common components among of the user interface of different platforms. There may be some components which are uncommon between both platforms and do not have any alternate in another platform but these are skipped.

3.2 Automatic Test Cases Generation and Execution Problem

Analyzing GUI of mobile apps and then generating and executing test cases is very difficult if done manually. Different techniques are proposed to overcome this issue but all these techniques have some limitations. PATS [13] is just applicable for android apps. In PATS there is no proper scheduling process of assigning user interfaces to slave nodes and node failure issue is also not considered in this framework. Technique of ReWeb and TestWeb [14] are proposed again only for android applications and not practically implemented yet. Fuzz Testing [15] floods AuTs GUI with infeasible events and also cannot certify effective failure revealing. For R and R [15] traces script is required for each device, because traces scripts are often coupled to locations in the screen. In T+ [15] encapsulated components (for example, KeyboardView, AutoCompleteTextView, CalendarView, DatePi cker) cannot be analyzed during systematic exploration because the subcomponents (for example, each key of the keyboard) are not available for ripping at execution time. AndroidRipper [16] is used to generate test cases automatically on the basis of events analyzed. But these test cases do not guarantee complete and efficient testing and bug finding in AuT because these test cases do not contain perturbed inputs.

3.3 Test Cases Migration Problem

Generating different test cases for the same app based on different platforms is time consuming and costly. To use test cases of app of one platform over the other, test cases migration technique [17] is proposed. But this not yet implemented and also not fully automated. Action matching method in this technique is manual, which will take a lot of time and will make this technique useless even if implemented. It is also not discussed for the actions which remains unmatched what technique will be followed for the generation of test cases for these actions.

3.4 Device Selection Problem

Proper device selection for testing android applications is an issue. To solve this problem Combinatorial Approach [18] was proposed but limitation of this approach is that selection of devices for testing mobile apps is manual, which is much time consuming task. No tool is there for automated selection of devices.

3.5 Enabling A/B Testing Problem

A/B testing is comparing two versions of application by distributing it amongst users and analyze their behavior. Remote User

interface exchange [19] was proposed to enable A/B testing for mobile apps. The issue in this technique is that the user behavior of different user groups using different user interfaces cannot be analyzed which is the main theme of A/B testing. Only remote modification of user interface functionality is discussed. Experience sampling method (ESM) [22] is used to evaluate mobile apps by capturing users behavior. In ESM the data recorded from the users might be unreliable because there is no proper procedure to check reliability of the data.

3.6 Problems in Model Based Gui Testing Techniques

Model based testing is a technique in which the test cases are generated from a model that describes the functional aspects of the system under test. Creation of a model to enable model based testing is issue. Different techniques are introduced [20] [21] [25] [6] [26] enabling GUI modeling and model based testing but all these techniques have limitations. Optical character recognition (OCR) ripper [20] is not fully automatic as GUI ripper do not capture system interaction events. Also OCR gives inaccurate results and misses GUI windows. No specific evaluation method to check the completeness of GUI specified by MIM [21]. There is no tool to support MIM language nor any mechanism to integrate MIM with MDD. Pattern based GUI testing [23] [24] is complicated approach for testing GUI of mobile apps. This approach do not detect and test all the events (like swipe, long press) on the GUI of the mobile app. At different screen sizes the PBGT (Pattern Based GUI Testing) cannot detect the exact location of the element hence effects testing results. Also PBGT does not test other platforms (IOS, Windows, etc) except Android. In ADAutomation technique [25] user behavior model is extracted from activity diagram. Test cases generated on the basis activity diagrams cannot give accurate results. Activity diagram provides the flow of activities hence cannot analyze all GUI components and actions taken against those components (i.e. selection based on radio buttons or checkboxes etc.). In [6] combination of two tools SlumDroid and GUIAnalyzer are used to produce GUI model of AuT and produce perturbed inputs. These tools do not test performance and compatibility of apps.

4. Approaches to Mobile Gui Testing

The following sections provide an overview of the different approaches developed to test mobile GUI. These approaches identify three main open areas for obile GUI testing, and limitations linked with each approaches regarding automated mobile GUI application testing.

4.1 Test Cases Generation and Execution through Gui Analysis

The critical issues in the automation of mobile GUI testing are to efficiently analyze the GUI of a mobile app, autogenerating test cases, and execution of those test cases. Researchers have proposed different approaches, such as research in [13][17] and [16] have proposed solutions to automatically analyze GUI for generating and executing test cases. PATS [13] master node assigns user interfaces to slave nodes for analysis and then generates test cases. Efficiently analyzing all the components of application even with the devices having different screen sizes and generation of actionable test cases is the key performance indicator of this approach. Figure 1 is one such model. Current tools and techniques based on this approach have many shortcomings in this direction.

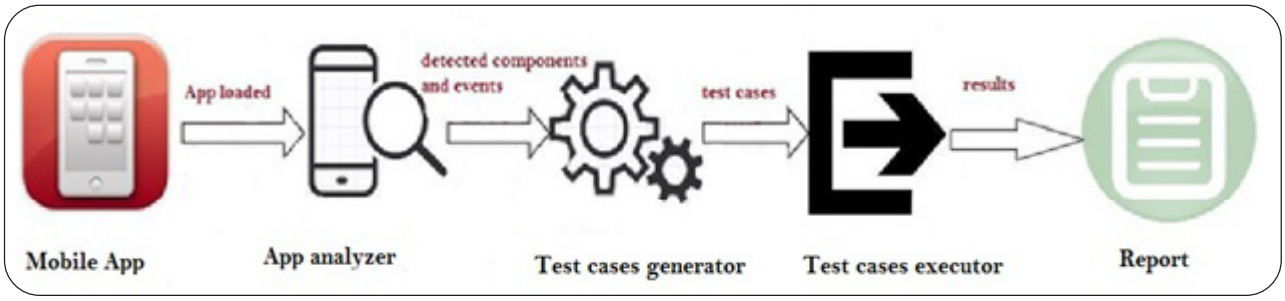


Figure 1. Basic model for testing through GUI analyzing

4.2 Model Based Testing

Model based testing technique is also used for testing mobile apps. In model based testing test cases are easily generated on the basis of created model, as compared with other techniques approaches. In this technique a formal model of an app is created. For example, 2 presents a model for a simple “hello world” pronunciation app. The model may be developed manually by the

developers or automatically by analyzing different GUI events of the AuT. Model may be created from the activity diagrams or requirements specifications [25]. The model describes the app at a level of abstraction necessary for automatic generation of test cases. Generated test cases are automatically executed, evaluated and documented. Many different techniques have been presented which support model based testing of mobile app, for example, [14] [15] [20] [21] [25] [6] and [26]. Each model based testing technique follows different approach. Such as OCR [20] uses the technique of image processing to extract character information such as clickable widgets in windows. MIM [21] specifies all the characteristics of final UI of an application. In ADAutomation [25] UML activity diagrams are used to create model of application. SlumDroid [6] uses GUI Ripping and creation of navigation model of AuT. Although these techniques work well in certain directions but still not adequate for testing all aspects of mobile GUI testing. OCR is not fully automated as it is not able to capture system interaction events like flip and also misses GUI windows. There is no tool to support MIM language nor any mechanism to integrate MIM with MDD. These limitation in MIM make it impractical as does not allow for model driven development and testing. In ADAutomation test cases generated from activity diagrams cannot give accurate results. As in this technique model is extracted from activity diagrams which just provide activity flow and does not specify all GUI components. Along with these approaches other techniques also have limitations such as improper error reporting procedure and platform dependency (limited to android), hence failed to prove the effectiveness of model base GUI testing of mobile apps.

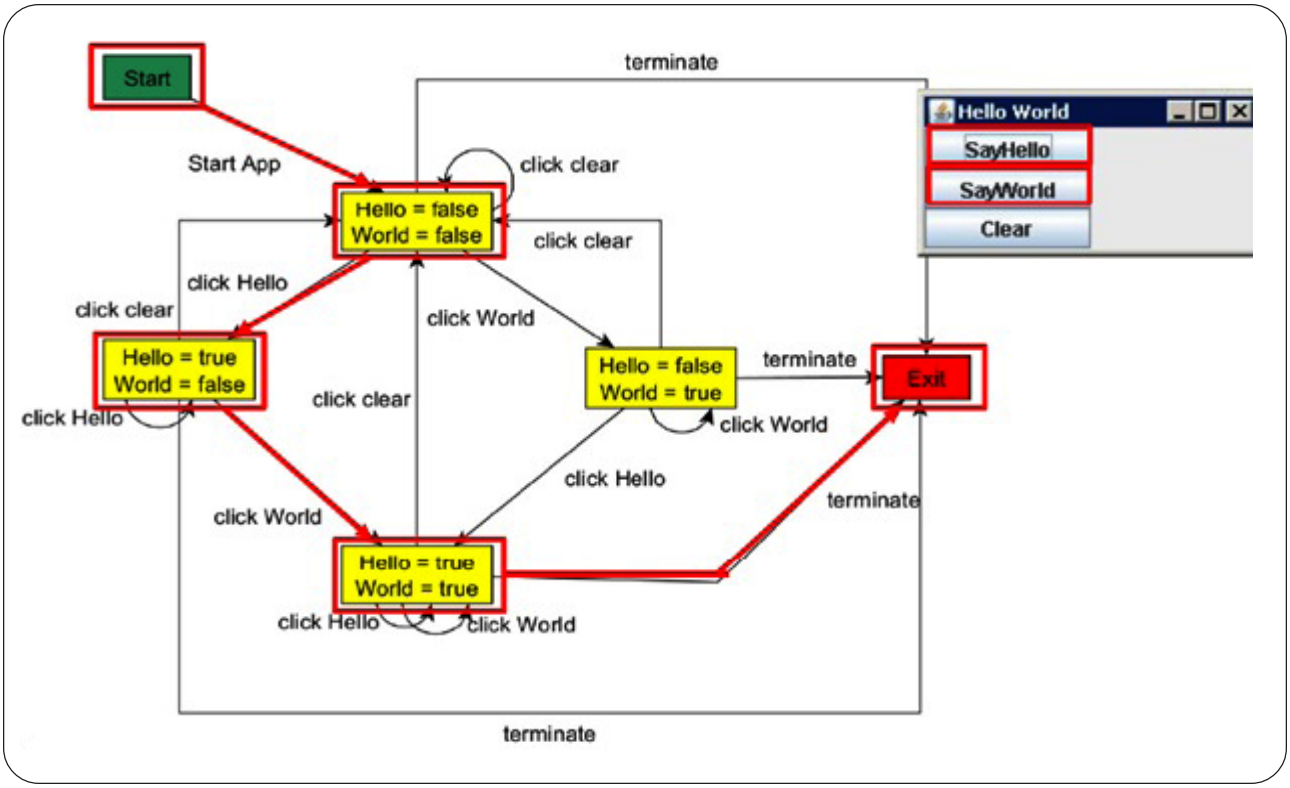


Figure 2. Transition-Based (FSM) Model of hello world pronunciation application

4.3 Cloud Based Testing

The rapid growth in the use of mobile devices is matched with an increase in development of mobile apps [3]. Testing mobile apps is more challenging than traditional software because of diverse platforms and versions of operating systems of mobile devices. To check the compatibility of GUI of mobile app across different platforms and devices simultaneously is a problem for researchers. To overcome this issue cloud based testing techniques are introduced [5-9]. The basic model of cloud testing is shown in Figure 3. All these techniques put an effort towards testing mobile app across different platforms and devices taking the advantage of cloud computing. But these techniques fails to achieve complete GUI testing against different platforms as analyzing uncommon components of application of different platforms, perform load testing and recording GUI errors are still persisting issues in cloud base testing.

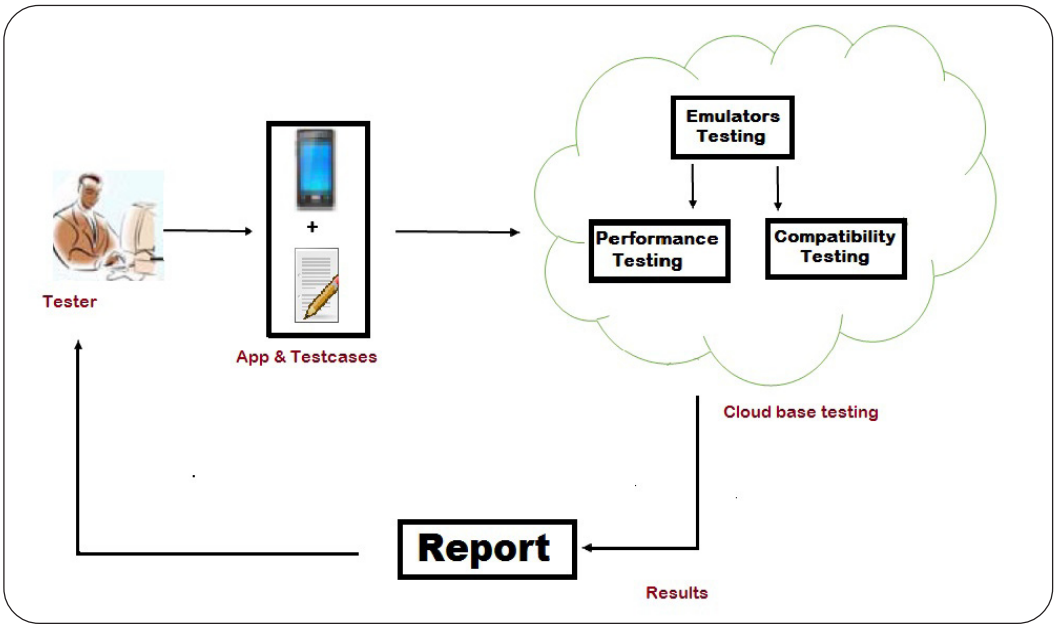


Figure 3. Basic cloud based testing model

5. Selection of an Appropriate Testing Approach

The previous section defines three main approaches for automatic mobile GUI testing. In GUI analysis based approach test cases are automatically generated through GUI evaluation. This approach does not provide complete test coverage and functionality testing of an application as it works on analyzing GUI and does not produce test cases for missing GUI components. Cloud based testing approach provides performance and reliability of an application by running application on different devices and emulators. In this approach no methodology for complete functionality testing and maximum testing coverage is specified. In model based testing (MBT) approach test cases are automatically generated from the functional model of an application through which complete functionality testing and maximum test coverage is achieved. Table 1 shows comparison between these three main testing approaches.

Although MBT is best for automatic functionality testing based on model generated through GUI navigation but it has some critical issues relating to it. Creation of a proper functional model of AuT is an issue as no standard approach is there for creating the model. There are limited tools and skills support available for MBT and the available tools are not much user friendly. MBT focuses on the functional testing of AuT and disregards performance, security, and compatibility testing [27] [28], and cannot represent user behavior [29]. All these limitation hinder global acceptance of MBT.

6. Functionality-performance-compatibility Model

Functionality, performance and compatibility are the main factors of acceptance for a mobile application. In TriTest, for functional testing a model of the Application under Test (AuT) is created with GUI Ripping Technique. This model is used by GUIAnalyzer to produce test cases automatically. These test cases are executed to test functionality of AuT.

After completion of functional testing, performance and compatibility testing will be started, because in our framework performance and compatibility testing will use results of functional testing process. Performance is the measure of how efficiently an application respond to users. Each application defines their own performance and compatibility requirements according to which these applications have to be tested. But we have selected set of those requirements that are common for all applications. Sometimes an application may fail to describe a performance attribute which is a critical issue for such type of application. Therefore, we are proposing set of attributes for performance testing which should be a bench mark criteria for performance testing of an application. These performance attributes are selected because they are common for all types of application [11]. These are Response time, Throughput, and Resources consumption.

Response time is the time taken by an application to response back to the user after giving input. To integrate performance testing in our MBT framework the response time is measured by calculating the time taken by application to respond when test cases starts executing. Throughput is total number of events processed in given amount of time. It is measured by number of test cases executed in a given amount of time. Resource utilization by an application such as memory and processor consumption also have a direct effect on the performance of an application. If an application consumes more memory and processor time, it will affect the performance of device. Resources (Memory and processor) consumption by an application can be measured when the application is operational by executing functional test cases, then total memory and processor consumed can be calculated.

Compatibility testing will be done of most commonly used real world devices or emulators. Compatibility testing is done based on compatibility testing model based on ISO/IEC9126- 1. Compatibility of an application will be assessed by measuring adaptability, installability, and replaceability of application over that devices as these three are considered as sub characteristics of compatibility in ISO/IEC 9126-1 [7]. Installabiltiy and adaptability results will be taken form functional testing process. As during functional testing process the application is automatically installed on android device / emulator for executing test cases on it. As adoptability can be measured from hardware and operating system adoptability [7]. If during functional testing process, the apk is installed successfully on android device / emulator and test cases are executed successfully then this application is adaptable. Replaceability of an application will be measured separately after successful measurement of installability and adaptability as replacibility testing is performed when there is some modifications in application. For replaceability testing, the application will be retested and its results will be compared with previous testing results. Figure 4 delineate architecture of the approach having model based functionality, performance and compatibility testing.

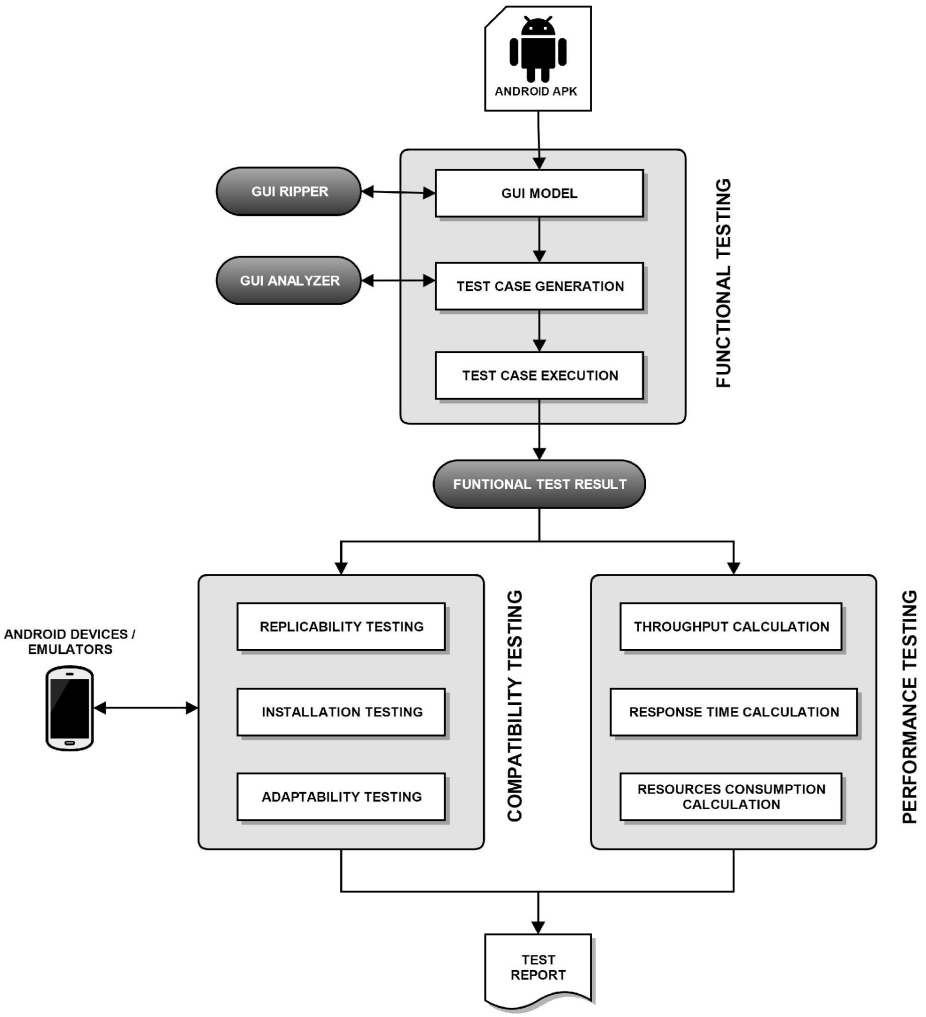


Figure 4. Model based testing architecture with functionality, performance and compatibility testing

GUI Testing Approaches	Test Coverage	Functionality Testing	Performance Testing	Automicity
UI Analysing Based	Partial	Partial	Nil	Full
Model Based	Full	Full	Nil	Full
Cloud Based	Partial	Partial	Full	Partial

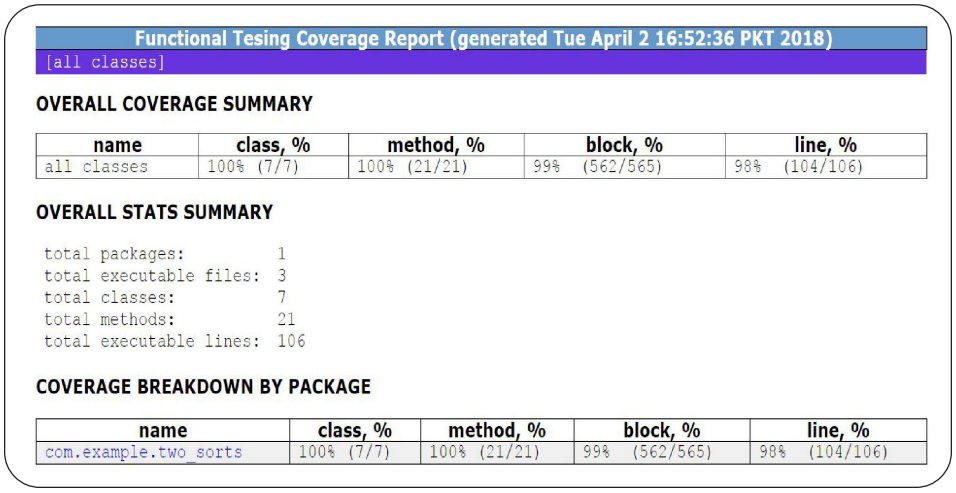
Table 1. Comaprison of gui testing approaches

7. Results

At the end of the testing process one major report is generated in HTML format which have four major portions. First portion is functionality testing report as shown in figure 4, 7. It contains the complete functionality testing coverage report of the application. This report contains total percent of classes, methods, blocks and lines covered in the testing process.

Second portion is the performance testing report as shown in figure 4.8. It contains response time, throughput, total memory, and processor consumed by the application.

Third portion is compatibility testing report of an application. It contains compatibility summary if an application is successfully installed and is adaptable to hardware and operating system of the device. This report also contains the supported API level by the application.



Functional Tesing Coverage Report (generated Tue April 2 16:52:36 PKT 2018)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	100% (7/7)	100% (21/21)	99% (562/565)	98% (104/106)

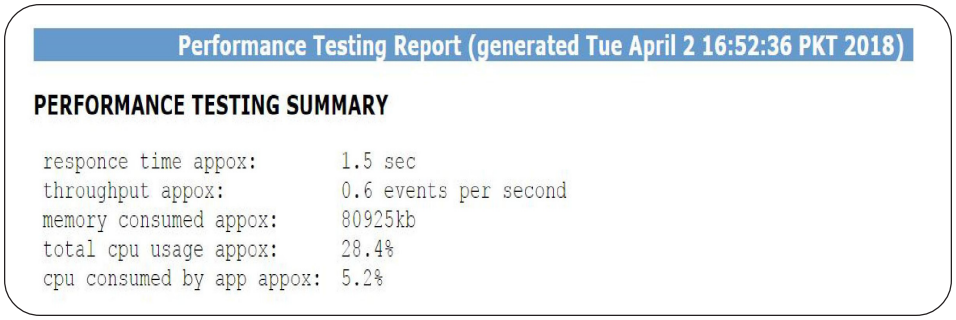
OVERALL STATS SUMMARY

total packages: 1
total executable files: 3
total classes: 7
total methods: 21
total executable lines: 106

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
com.example.two_sorts	100% (7/7)	100% (21/21)	99% (562/565)	98% (104/106)

Figure 5. Functionality testing coverage report



Performance Testing Report (generated Tue April 2 16:52:36 PKT 2018)

PERFORMANCE TESTING SUMMARY

responce time appox: 1.5 sec
throughput appox: 0.6 events per second
memory consumed appox: 80925kb
total cpu usage appox: 28.4%
cpu consumed by app appox: 5.2%

Figure 6. Performance testing report

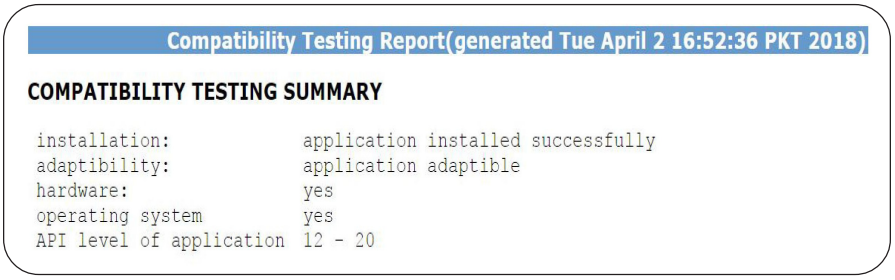


Figure 7. Compatibility testing report

7. Conclusion

Acceptance of mobile apps is dependent on simple, interactive, and user friendly GUI of applications. This paper presented a review on the existing GUI testing techniques of mobile apps. The paper outlines benefits and limitations of different GUI testing techniques as well as critical issues regarding GUI testing of mobile apps are focused. TriTest, a model based approach is proposed, which aims to automatically test functionality, compatibility, and performance of mobile apps. In mobile apps testing analyzing GUI, test cases generation, test cases execution, developing formal model, cross platform testing, and cloud based testing are major issues for which different techniques and tools are presented. Model based testing differs from ordinary testing techniques by following proper model for GUI testing. Still there are issues with MBT like creation of proper functional model. Currently MBT also do not support performance, compatibility, security, and usability testing. There is a need to handle these factors by facilitating performance and compatibility testing with in the model. The proposed TriTest framework incorporates all these critical app quality attributes and aims to helps the user in achieving efficient automated mobile app GUI test case generation and execution.

References

[1] Rivera, J., Van Der Meulen, R. (2015). Gartner says worldwide smartphone sales recorded slowest growth rate since 2013. Gartner. Accessed: 2017-05-28. [Online]. Available: http://www.gartner.com/newsroom/id/3115517. (August).

[2] (2017) Mobile/tablet operating system market share. Net Applications. Accessed: 2017-07-22. [Online]. Available: https://www.netmarketshare.com/operating-system-marketshare. aspx?qprid=8&qpcustomd=1

[3] Van Der Meulen, R., Rivera, J. (2014). Gartner says annual smartphone sales surpassed sales of feature phones for the first time in 2013. Gartner. Accessed: 2016-04-29. [Online]. Available: http://www.gartner.com/newsroom/id/2665715. (February).

[4] Santos, A., Correia, I. (2015). Mobile testing in software industry using agile: Challenges and opportunities, *In*: *Software Testing, Verification and Validation (ICST)*, 2015 IEEE 8th International Conference on. IEEE, p. 1–2.

[5] Cormier, J. (2010). Over 75 percent of mobile apps unused. Accessed: 2016-04-29. [Online]. Available: http://www.examiner.com/article/study-over-75-percent-of-mobileapps- unused. (September).

[6] Imparato, G. (2015). A combined technique of gui ripping and input perturbation testing for android apps, *In*: *Proceedings of the 37th International Conference on Software Engineering*-Volume 2. IEEE Press, p. 760–762.

[7] Liu, Z., Hu, Y., Cai, L. (2014). Research on software security and compatibility test for mobile application, *In*: *Innovative Computing Technology* (*INTECH*), 2014 Fourth International Conference on. IEEE, p. 140–145.

[8] Baride, S., Dutta, K. (2011). A cloud based software testing paradigm for mobile applications, *ACM SIGSOFT Software Engineering Notes*, 36 (3) 1–4, 2011.

[9] Malek, S., Esfahani, N., Kacem, T., Mahmood, R., Mirzaei, N., Stavrou, A. (2012). A framework for automated security testing of android applications on the cloud, *In: Software Security and Reliability Companion (SERE-C)*, 2012 IEEE Sixth International Conference on. IEEE, p. 35–36.

[10] Murugesan, L., Balasubramanian, P. (2014). Cloud based mobile application testing, *In*: *Computer and Information Science (ICIS)*, 2014 IEEE/ACIS 13th International Conference on. IEEE, p. 287–289.

[11] Prathibhan, C. M., Malini, A., Venkatesh, N., Sundarakantham, K. (2014). An automated testing framework for testing android mobile applications in the cloud, *In Advanced Communication Control and Computing Technologies* (*ICACCCT*), 2014 International Conference on. IEEE, p. 1216–1219.

[12] Bayley, I., Flood, D., Harrison, R., Martin, C. (2012). Mobitest: a crossplatform tool for testing mobile applications.

[13] Wen, H. L., Lin, C. H., Hsieh, T. H., Yang, C. Z. (2015). Pats: A parallel gui testing framework for android applications, *In*: *Computer Software and Applications Conference* (*COMPSAC*), 2015 IEEE 39th Annual, vol. 2. IEEE, p. 210–215.

[14] Ahmed, M., Ibrahim, R., Ibrahim, N. (2015). Adaptation model for testing android application, *In*: Computing Technology and Information Management (ICCTIM), 2015 Second International Conference on. IEEE, p. 130–133.

[15] Linares-V´asquez, M. (2015). Enabling testing of android apps, *In*: Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, p. 763–765.

[16] Amalfitano, D., Fasolino, A. R., Tramontana, P., De Carmine, S., Memon, A. M. (2012). Using gui ripping for automated testing of android applications, *In*: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM, p. 258– 261.

[17] Choudhary, Roy. S. (2014). Cross-platform testing and maintenance of web and mobile applications, *In*: Companion Proceedings of the 36th International Conference on Software Engineering. ACM, p. 642–645.

[18] Vilkomir, S., Amstutz, B. (2014). Using combinatorial approaches for testing mobile applications, *In*: Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on. IEEE, p. 78–83.

[19] Lettner, F., Holzmann, C., Hutflesz, P. (2013). Enabling a/b testing of native mobile applications by remote user interface exchange, In Computer Aided Systems Theory-EUROCAST 2013. Springer, p. 458– 466.

[20] Wu, Y., Liu, Z. (2012). A model based testing approach for mobile device, *In*: Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on. IEEE, p. 1885–1888.

[21] Geiger-Prat, S., Marin, B., Espana, S., Giachetti, G. (2015). A GUI modeling language for mobile applications, *In*: Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on. IEEE, p. 76–87.

[22] Lee, M., Kim, G. J. (2015). On applying experience sampling method to a/b testing of mobile applications: A case study, *In*: Human-Computer Interaction–INTERACT 2015. Springer, p. 203–210.

[23] Costa, P., Paiva, A. C., Nabuco, M. (2014). Pattern based gui testing for mobile applications, *In*: Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the. IEEE, p. 66–74.

[24] Moreira, R. M., Paiva, A. C. (2014). Pbgt tool: an integrated modeling and testing environment for pattern-based gui testing, *In*: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ACM, p. 863–866.

[25] Li, A., Qin, Z., Chen, M., Liu, J. (2014). Adautomation: An activity diagram based automated gui testing framework for smartphone applications, *In*: Software Security and Reliability (SERE), 2014 Eighth International Conference on. IEEE, p. 68–77.

[26] Benouda, H., Azizi, M., Esbai, R., Moussaoui, M. (2016). Mda approach to automate code generation for mobile applications, *In*: Mobile and Wireless Technologies 2016. Springer, p. 241–250.

[27] Binder, R. V., Legeard, B., Kramer, A. (2015). Model-based testing: where does it stand? *Communications of the ACM*, 58 (2).

[28] Binder, R. (2011). Model-based testing user survey: Results and analysis, System Verification Associates.

[29] Dias Neto, A. C., Subramanyan, R., Vieira, M., Travassos, G. H. (2007). A survey on model-based testing approaches: a systematic review, *In*: Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007. ACM, *p*. 31–36.