Received: 13 September 2017 | Revised: 12 May 2018 | Accepted: 15 May 2018

DOI: 10.1002/smr.1963

RESEARCH ARTICLE - METHODOLOGY

WILEY Journal of Software: Evolution and Process

Automated parallel GUI testing as a service for mobile applications

Amira Ali | Huda Amin Maghawry | Nagwa Badr

Department of Information Systems, Ain Shams University, Cairo, Egypt

Correspondence
Amira Ali, Department of Information Systems, Ain Shams University, Cairo, Egypt.
Email: amira.aly92@live.com

Abstract

Recently, testing mobile applications is gaining much attention due to the widespread of smartphones and the tremendous number of mobile applications development. It is essential to test mobile applications before being released for the public use. Graphical user interface (GUI) testing is a type of mobile applications testing conducted to ensure the proper functionality of the GUI components. Typically, GUI testing requires a lot of effort and time whether manual or automatic. Cloud computing is an emerging technology that can be used in the software engineering field to overcome the defects of the traditional testing approaches by using cloud computing resources. As a result, testing-as-a-service is introduced as a service model that conducts all testing activities in a fully automated manner. In this paper, a system for mobile applications GUI testing based on testing-as-a-service architecture is proposed. The proposed system performs all testing activities including automatic test case generation and simultaneous test execution on multiple virtual nodes for testing Android-based applications. The proposed system reduces testing time and meets fast time-to market constraint of mobile applications. Moreover, the proposed system architecture addresses many issues such as maximizing resource utilization, continuous monitoring to ensure system reliability, and applying fault-tolerance approach to handle occurrence of any failure.

KEYWORDS

TaaS, cloud computing, mobile application testing, GUI testing, Appium

1 | INTRODUCTION

Testing mobile applications is defined as the process of testing functionality and quality of the software applications developed for smartphones.[1] Nowadays, mobile applications have gained much popularity due to the continuous increase in the number of smartphone users. Thousands of mobile applications are developed daily, and these applications are available in many stores giving users plenty of options to choose from them.[2] However, the majority of mobile applications are developed within restricted resources, budget, and limited time due to time-to-market pressure. This severely affects the quality of the mobile applications development.[3] Therefore, testing mobile applications adequately within a constricted time and cost becomes a big challenge that will affect the continuality of the mobile applications development.[4] Mobile applications testing requires much time, cost, resources, and skillful tester and tools for writing and executing test cases.[5-7]

In the last few years, cloud computing has gained significant attention. Cloud computing is the next revolution of distributed computing paradigm which can support on-demand service sharing with higher level of flexibility and dynamic scalability to sharable computing resources which are configurable[8,9] Thus, migrating software testing to the cloud environment and leveraging cloud unlimited storage, resources (hardware and software), and scalable infrastructure is the best solution to overcome difficulties of the traditional testing strategies.[10-15] In the cloud-based environment, there is no need to set up test environment resources due to the existence of readily on-demand virtual test environment. This helps to

meet fast time-to-market constraint of the mobile applications. As a result, testing-as-a-service (TaaS) is presented as a service model that automatically conducts all testing activities including automatic test cases generation, test execution, and test report generation.[16-18] The application under test (AUT) is submitted to the TaaS platform. The TaaS platform automatically accomplishes all the test activities, generates a detailed test report, and submits the test report to the user.

In this paper, a system for mobile applications graphical user interface (GUI) testing based on TaaS architecture is presented. The proposed system has scalable and easy-to-use architecture. The critical factors and challenges that lead to efficient and cost-effective testing for mobile applications are addressed. GUI becomes pervasive for interacting with mobile applications. Therefore, users of mobile applications care that the GUI controls of the mobile applications are functioning properly and working according to the required specifications.[19] GUI testing for mobile applications, as a consolidated type of testing, aims to ensure that AUT meets its functional requirements and achieves high quality standard. GUI test cases are composed of sequences of input events to ensure that GUI components conform to their required specifications. Model-Based Testing (MBT) is defined as a type of testing methodology that derives test cases from different models. Models has 2 types: system requirements models (black box testing) and models constructed from source code (white box testing).[20,21]

GUI test cases generation requires accurate modeling to the user behaviors to imitate the interactions between users of mobile applications and the GUI controls. In this paper, test case generation approach based on UML activity diagram is introduced. The Unified Modeling Language (UML) is a widely used modeling language that effectively model different software artifacts.[22] UML activity diagram is used to model software workflows in terms of a stepwise activities and actions. Activity diagram becomes a good candidate for representing user behavior which allows automated GUI mobile applications testing.[23]

In practice, GUI testing takes too much time which may lead to delay-to-market for mobile applications. That is why it is essential to develop automated cost-effective testing techniques as an alternative. Due to the rapid growth of Android applications,[24] the proposed system focuses on GUI testing of Android applications in a cloud-based environment. Besides, Android is an open source platform which facilitates availability of Android applications for our experiments.

The rest of the paper is organized as follows: Section 2 highlights the most relevant related work for testing mobile applications in a cloud-based environment. Section 3 introduces the proposed system architecture and a detailed description of the functionality of its modules. Section 4 presents experimental results of the proposed system. Finally, Section 5 introduces conclusion and future work.

2 | RELATED WORK

Recently, several researches concerned with studying mobile applications testing in a cloud-based environment from different perspectives. Some researches provide informative discussion to mobile TaaS in terms of definition, requirements, benefits, issues, and challenges as in the literatures.[25-28]

Furthermore, many researchers introduced prototype architecture to the mobile TaaS. For example, Shenbin Zhang et al[29] presented a mobile app functional test solution for Android native app based on TaaS platform. User is allowed to customize the test environment by configuring the devices' context parameters such as location, language, and network speed and so on. Besides, authors introduced functional traversal approach for test script generation. The functional traversal approach starts with calculating weight of each operation based on the operation features as control's type, size, text, and the action of operation. Then, operations with high weight values are executed. The authors focused on traversing activity that includes operations whose weight are more than certain threshold value otherwise the activity will not be accessed. This leads to incomplete coverage (ie, activity coverage, operation coverage) of the generated test scripts.

Riyadh Mahmood et al[30] presented a framework for automated security testing for Android applications in the cloud environment. Numerous of heuristics are used in generating test cases to raise likelihood of discovering security vulnerabilities in Android app.

Tao Zhang et al[31] proposed a compatibility testing service for mobile applications. A mobile compatibility test method that generates compatibility test sequences is introduced. It is based on a feature tree to model compatibility features. Each leaf node in the feature tree represents a basic compatibility feature. K-Means algorithm is used to cluster mobile devices with similar compatibility features. Then, device clusters are ranked according to their market share. In addition, the authors discussed the infrastructure required for mobile compatibility testing as well as the mobile compatibility test server, which is composed of 5 layers: (1) test tenant, (2) test service, (3) compatibility testing, (4) communication, and (5) test database.

Tor-Morten Grønli et al[32] presented a conceptual prototype for parallel cross-platform mobile test execution framework known as (Mobilette). However, Mobilette can test basic mobile applications only because it supports few basic GUI actions. Besides, Mobilette framework only performs test execution without concerning with the rest of testing activities especially test case generation, which is the core phase in the testing process.

Oleksii Starov et al[33] provided a comprehensive view to the Cloud Testing of Mobile Systems (CTOMS). CTOMS is a distributed TaaS platform for mobile development. The presented system has the following layers: presentation layer, platform layer, and the information logical layers of the cloud solution. The master application is used to represent the platform layer, for example: Google App Engine cloud and optional Hadoop instances layer that leverages the MapReduce algorithm to distribute test between nodes and gather the results. The master application provides the presentation to the end-user and to the slave node computers. The information layer is represented by the cloud data storage. The

author presented an overall architecture to CTOMS. However, the techniques used to carry out each test activity involved in the testing process are not discussed.

There are many commercial cloud-based mobile testing tools and services like: Xamarin Test Cloud and Perfecto Test. Xamarin Test Cloud[34] provides many testing tools for cross platform mobile applications testing and provides thousands of devices to run test cases. Xamarin Test Cloud has limited access to open source libraries. Perfecto Test[35] allows real-time testing of a mobile application on different platforms (eg, Android, iOS). Despite the advantages of Xamarin Test Cloud and Perfecto Test in test execution, both tools do not perform automatic test cases generation.

There are several studies concerned with automatic test cases generation based on UML diagrams for testing mobile application. Ang Li et al[36] presented a framework for mobile applications testing called (ADAutomation). ADAutomation framework supports user behavior modeling, test case generation, test execution, and post-test analysis and debugging. The authors introduced a test cases generation approach based on UML activity diagram. Anbunathan R et al[37] introduced a method to generate test cases after parsing sequence diagram then generating XML-based test cases and subsequently APK-based test scripts for Android mobiles. Virtual Test Engineer is a tool developed based on this method. Virtual Test Engineer is used for testing several Android applications. However, the previous 2 frameworks have a long overall completion time (ie, including test case generation and test execution) due to sequential test execution on 1 machine.

Therefore, there is a need for an integrated scalable system based on TaaS architecture. A system that can maximize utilization of cloud resources efficiently automate testing process and guarantee high test coverage.

3 | THE PROPOSED ARCHITECTURE

In this paper, an architecture for GUI mobile applications TaaS is proposed. User can submit a request to the proposed TaaS where the entire testing process will be carried out automatically and efficiently. Figure 1 shows the overall architecture of the proposed GUI mobile applications TaaS. The 3 layers of the proposed architecture are as follows: (1) user interface layer, (2) service management and testing layer, and (3) Database and Infrastructure as a Service (IaaS) layer.

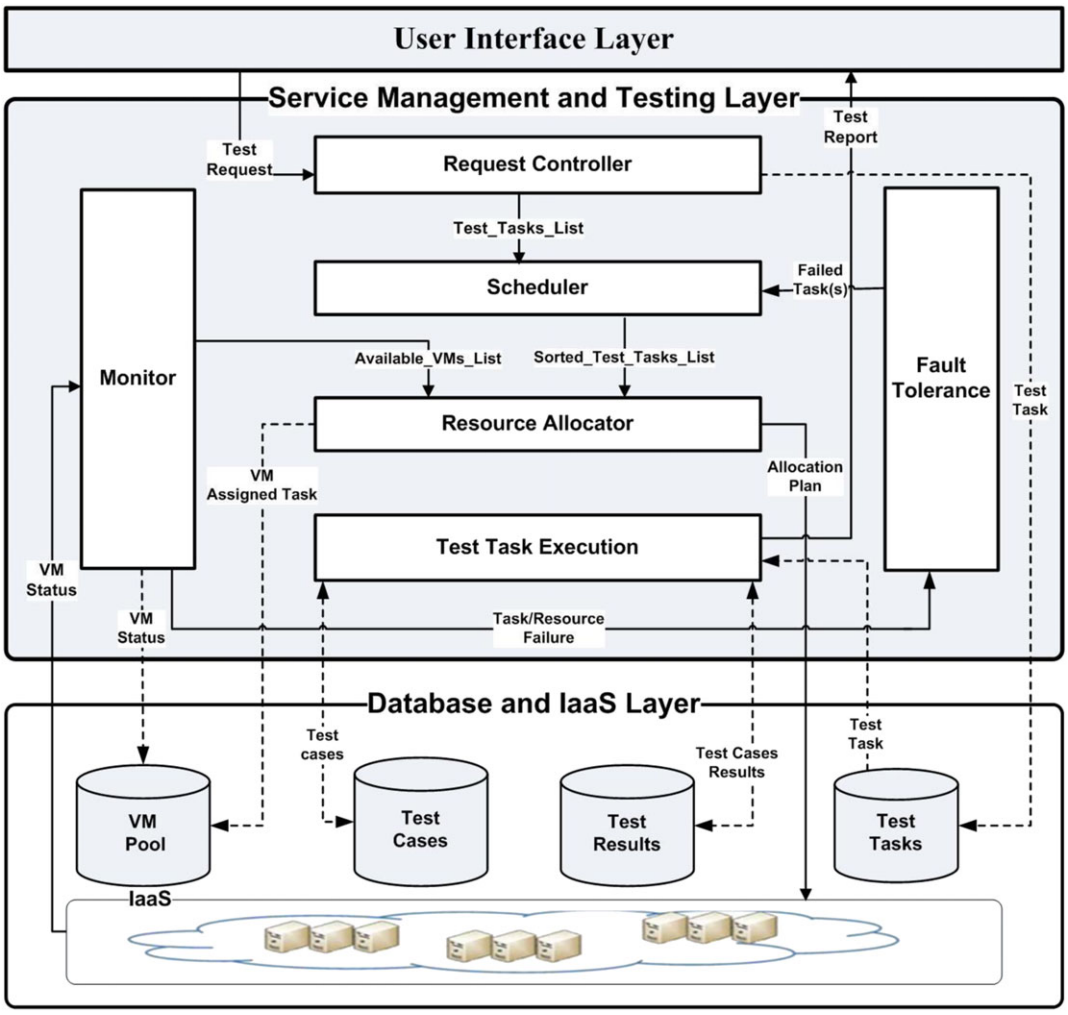

FIGURE 1 Mobile apps GUI TaaS architecture

A. User interface layer:

It is the first layer in the proposed architecture. User interface layer is a web front end that allows users to interact with the proposed TaaS architecture via the internet. Where users submit test request and define request requirements such as request deadline (ie, final time to accomplish testing of AUT). Finally, users receive the test results report. Detailed description of the user input submitted to the proposed architecture will be introduced in the next section.

B. Service management and testing layer:

This layer includes the main functionalities of the proposed architecture. The main modules of the service management and testing layer are: Request Controller, Scheduler, Monitor, Resource Allocator, Fault Tolerance, and Test Task Execution module. Detailed description of each module will be introduced in the next section.

C. Database and IaaS layer:

This layer includes 4 repositories: (1) Test Tasks repository stores test requests with their deadline and input data required to complete testing process such as activity diagram of AUT which will be described later in details. (2) Test Cases repository stores the generated test cases. (3) Test Results repository stores results of the executed test cases. (4) Virtual Machine (VM) Pool stores information about each VM such as VM state, VM load, and assigned tasks. All the required resources are provided to the users through the virtualization technology. Therefore, IaaS layer exists in order to include VMs where test tasks execution takes place.

The dependency between components of the proposed architecture and the organization of these components into packages are shown in Figure 2. The dotted line between components and packages represents dependency relationship. The dependency relationship means that the functionality of 1 component depends on the functionality provided by another one. The dotted line towards a data repositories means storing the component output in a data repository. While the dotted line from a data repository means retrieving data required to accomplish a component functionality. Components in the proposed system architecture are organized into 2 main packages: service management package and test task execution package. There is a dependency relationship between the 2 packages, as test task execution package depends on the functionality

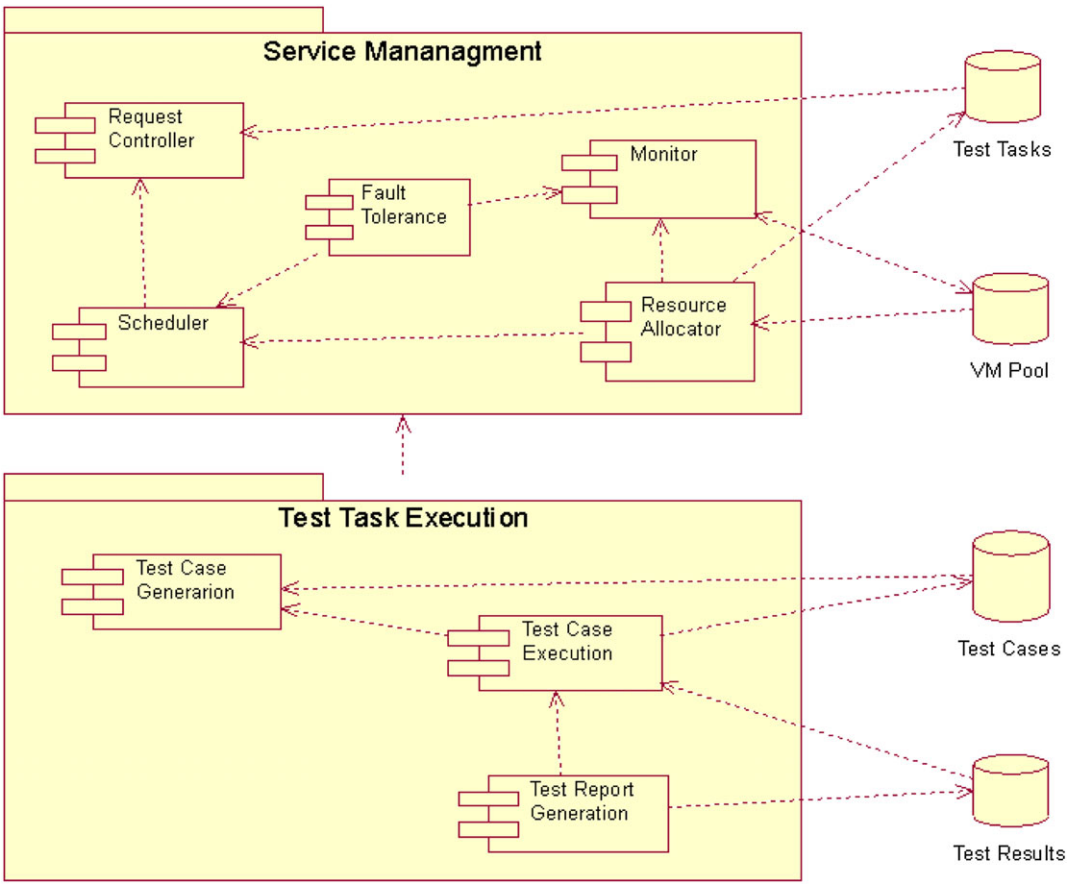

FIGURE 2 The proposed architecture component diagram

provided by the components included in the service management package. Test task execution package includes the following components: test case generation, test case execution, and test report generation. These components are responsible for carrying out the entire mobile applications GUI testing procedure. Service management package includes the following components: scheduler, monitor, resource allocator, fault tolerance, and request controller. The components of the service management package are responsible for managing test tasks and resources incorporated in the proposed system architecture. The functionality and the working approach of each component in the proposed system will be introduced in the next section.

3.1 | The service management and testing layer

This section presents an insightful description to each module in the service management and testing layer in the proposed architecture. The discussion of functionality of each module goes as follow.

A. Request controller module

It is the first module in the service management and testing layer. A user submits a test request to the proposed system through the user interface. The request controller coverts the test request into a test task and stores the task into Test Tasks repository. Then, the request controller module adds the task to the Test_Tasks_List.

B. Scheduler module

Scheduler module defines the execution order of the test tasks. In the proposed architecture, the scheduler module is implemented as a local service that applies a queuing method to determine the priority of each task in Test_Tasks_List. The scheduler receives Test_Tasks_List from the request controller. The queuing method considers both task deadline and waiting time to calculate the tasks priorities.[38] Task deadline is the final time defined by the user to accomplish testing of the AUT, while task's waiting time is the duration between current time and the time when the task was submitted by the user. Then, the scheduler module arranges tasks in descending order according to their priorities. Tasks are sorted in ascending order according to the task deadline (ie, a task with earlier deadline will have higher priority). Then, tasks having same deadline will be sorted according to their waiting time (ie, a task with higher waiting time will have higher priority).

C. Monitor module

To achieve higher level of system reliability, monitor module as a local service is used to follow the real-time status of all tasks and resources. It stores these information in the VM Pool as shown in Table. 1. The monitor module determines the Available_VMs_List by retrieving the state of each VM stored in the VM Pool. Then, the monitor module forwards the Available_VMs_List to the resource allocator module. Moreover, the monitor module detects failures, whether task failure or resource failure, then the failure details are forwarded to the fault tolerance module to manage it.

D. Resource allocator module

In a cloud-based environment, it is necessary to maximize the resource utilization and to improve system load balance.[39] Therefore, resource allocator module exists. Resource allocator module assigns task to the available virtual machines to execute the task in such a way to improve hardware and software resources utilization and guarantee load balance. After executing the task, its related resources will be released for another tasks. The resource allocator module receives Sorted_Test_Tasks_List and Available_VMs_List from scheduler and monitor modules, respectively. Then, the resource allocator module assigns tasks to VMs according to a dynamically generated allocation plan introduced in Figure 3.

As shown in Figure 3, the score of each VM is calculated according to the length of its waiting queue. Then, VMs are arranged in descending order according to the calculated score. The task with high priority is assigned to VMs with low score. In a cloud-based environment, test task can be decomposed into multiple sub tasks executed on multiple VMs in order to shorten the overall test time.[40] Therefore, test cases generated to test each AUT are executed concurrently on the assigned VMs.

TABLE 1 VM pool

VM ID	VM State	VM Load (number of assigned tasks)
1	ON	3
2	ON	5
3	OFF	0

```
Input: Sorted_Test_Tasks_List, Available_VMs_List
Output: VM Allocation Plan
Steps:
Start
For (int i=0 to n) // loop on all VMs in the Available_VMs_List
        VM_Score [ i ]= Get_WaitQueue_Length( Available_VMs_List [ i ] )
        //Wait Queue Length is number of tasks assigned to Available_VMs_List [ i ]
EndForLoop
Sorted_VMs [ ] = Sort_VMs_Descending (VM_Score [ ])
For each Task T in the Sorted_Test_Tasks_List
        VMs_ID [ ] =Get_VMs_with_Low_Score (VM_Score [ ])
        Add T and VM_ID [ ] in the VM Allocation Plan
        For (int i=0 to m) // loop on all VM_ID []
                Send T to VMs_ID [ i ]
        EndForLoop
EndForLoop
Return VM Allocation Plan
End
```

FIGURE 3 Allocation plan generation approach

E. Fault tolerance module

This module exists to handle failures that occur during the task execution. As mentioned before, the VMs real-time status and the tasks execution status are tracked by the monitor module to detect failures. The failure details captured by the monitor module are forwarded to the fault tolerance module. Fault tolerance module handles failures according to the following:

1. If the failure happens during the task execution. Then, the failed task is returned to the scheduler module to be rescheduled.
2. If the failure happens in a VM. Then, all the tasks assigned to the failed VM are returned to the scheduler module to be rescheduled.

F. Test task execution module

This module is responsible for performing the entire GUI testing process automatically for Android applications in a cloud-based environment. It aims to reduce the overall testing time and efforts by means of leveraging cloud resources. The module applies efficient testing approach to improve test quality and reliability.

Figure 4 shows Test Task Execution module. The proposed system receives input data required to automatically perform the entire testing process through the user interface layer. The input to Test Task Execution module are UML activity diagram, Android Application Package (APK), and GUI configurations file of AUT. UML diagrams are widely used in modeling application requirements. Thus, UML diagrams are considered one of the important sources for designing test cases.[22] Activity diagram can represent various control flows and dependency between GUI actions. That is why activity diagram becomes a good candidate for modeling user behavior to allow automated GUI testing of mobile applications.[36] APK is a Java bytecode packages are used to distribute and install Android applications. It allows testing mobile applications without requiring application source code. AUT configuration file includes GUI layout information as: GUI widget id, event required to fire GUI widget and widget type (eg, edit text or button).

The GUI configuration file of AUT is used to allow the proposed system to capture GUI interactions required for automatic test case generation and execution.

The process within the Test Task Execution module goes as follow:

- First, the XML Generator converts activity diagram drawn on a UML drawing tool (eg, Visual Paradigm[41]) into an XML-based activity diagram file. The obtained XML-based activity diagram file is the basic unit to establish the automated testing approach. The AUT activity diagram is converted to XML-based activity diagram file using XML parser tool (eg, Visual Paradigm[41]).

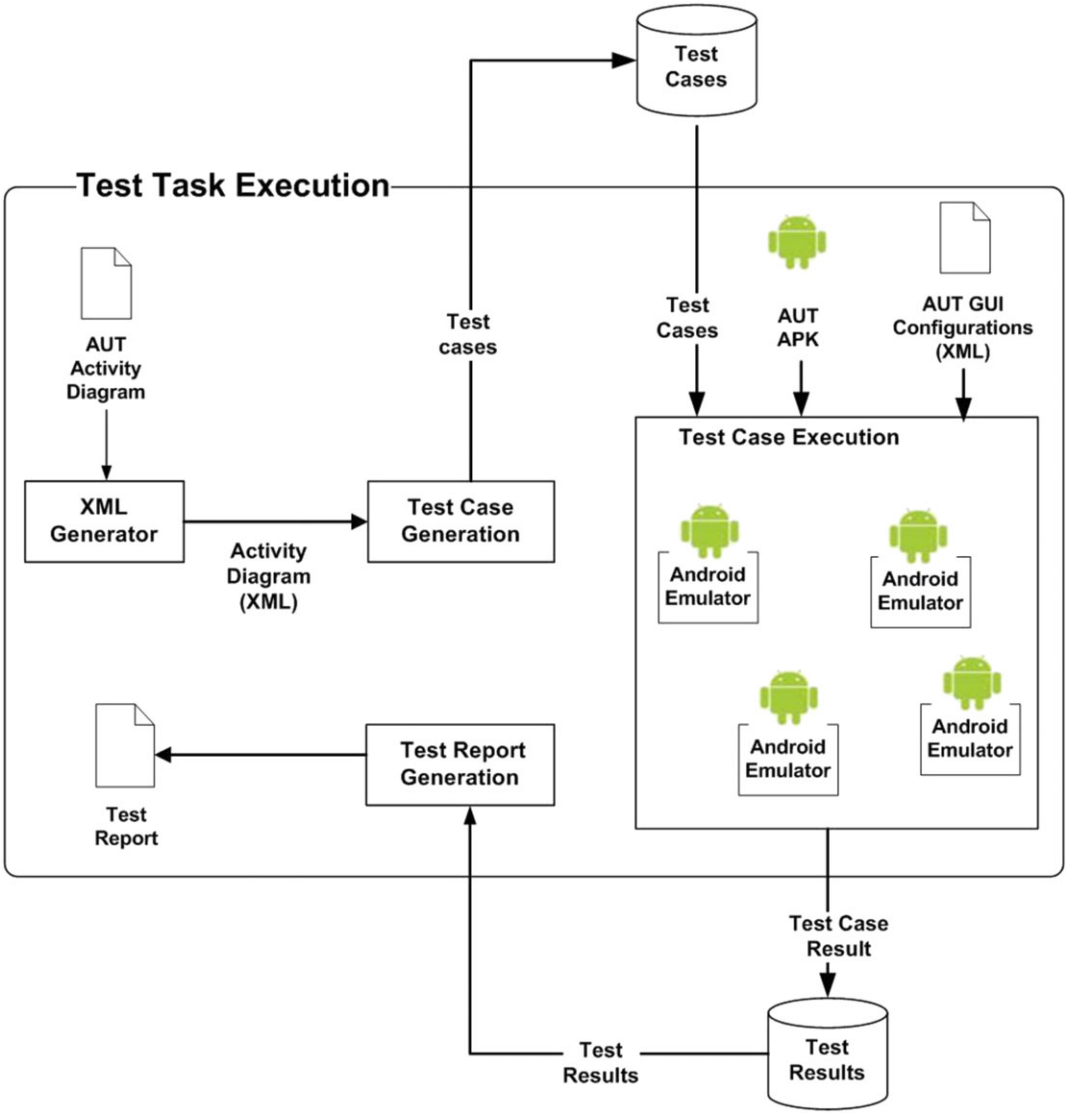

FIGURE 4 Test task execution module

- Test Case Generation module automatically generates a set of GUI test cases based on the information extracted from the obtained XML file. The generated test cases are stored in the Test Cases repository. Each test case consists of sequences of actions carried out on the GUI widgets of the AUT.
- Then, simultaneous test case execution takes place on multiple virtual nodes running Android Emulator in a cloud-based environment. Test Case Execution module installs APK of the AUT on Android Emulators before executing test cases, then executes test cases and stores the results of test cases in the Test Results repository.
- Finally, Test Report Generation module generates test report including results of the executed test cases and submit it to user.

Each sub module of Test Task Execution module will be explained in more details below:

i. Test Case Generation Module

Test case generation is the core phase in any testing process. Test cases should mimic a complete user behavior when using AUT. In GUI testing, each test case consists of a sequence of correlated actions carried out on GUI widgets of the AUT. Therefore, figuring out the dependency between the GUI actions is the key process in the GUI test case generation module. Mobile applications can be modeled using different types of models.[21,37,42-44] MBT is introduced as a feasible approach to test mobile applications in an effective and efficient way.[45] Therefore, GUI testing for android applications can be approached by MBT techniques, where formal model describing AUT at a level of details is required.

Figure 5 shows a dynamically automated approach for generating test case using activity diagram of the AUT after parsing it into an XML-based file. First, dependency table is generated to include all information extracted from the XML-based file. Each row in the dependency table represents element in the activity diagram. The generated dependency table has 4 columns:

1. First column includes element name.
2. Second column includes element type (ie, action, decision, initial, and final nodes).
3. Third column includes a list of all edges to this element.
4. Fourth column includes edges out from this element.

Edges are used to determine dependency between elements. The dependency between elements is defined by the presence of edge out from 1 element that is same as the edge to the current element.

Then, Finite-State Machine (FSM)[46] graph is generated based on the dependency table. Each node represents entry in the dependency table. Transition from 1 node to another is determined according to the dependency between elements defined in the dependency table. Initial and terminal nodes of the graph are the initial and final nodes of the activity diagram, respectively. Afterwards, the generated graph is traversed using depth first search algorithm[47] to find all the possible independent paths. Independent path[21] is defined as any path from the start node to the terminal node that introduces at least 1 new edge which has not been traversed before. Each path consists of sequence of correlated GUI actions that represents user behavior when using AUT.

Finally, Longest Common Subsequence algorithm (LCS)[48] is applied on the derived paths. LCS finds longest common sequences of GUI actions that always appear together. The reason behind applying LCS on the derived basis paths is to remove test paths that are included as a sub path in another path. This reduces the number of test paths as well as guarantees high coverage. Thus, each GUI component is exercised at least once. The obtained paths after applying LCS represent test cases.

ii. Test Case Execution Module

The generated test cases as well as AUT configurations file are used to obtain executable test scripts. Test scripts are executed using Appium.[49] Appium is a mobile automaton testing tool that has many advantages as: it is an open source tool, it does not need to modify AUT so no need to recompile AUT, and it supports scalability through running tests on multiple emulators.[50,51] There are many other mobile automaton testing tools like: Robotium and Calabash. While Robotium[52] can perform test execution just on 1 device at a time. Calabash[53] does not support complex test scenarios. That is why Appium is considered the best choice for the proposed system compared with the other mobile testing tools. In the proposed system, simultaneous test case execution takes place on multiple virtual nodes in a cloud-based environment, where Appium is installed on each virtual node running Android Emulator. Before executing test cases, APK of AUT is submitted to the assigned virtual nodes and installed on their Android Emulators.

iii. Test Report Generation Module

Finally, the proposed system generates test report and submits it to the user through the user interface. The test report includes test cases results gathered from multiple virtual nodes. The test report automatically organizes test results information. Test report includes steps of the automatically generated test cases and the result of execution of each test case whether failed or passed. This can be valuable for users to determine failed tests in order to resolve issues with the AUT.

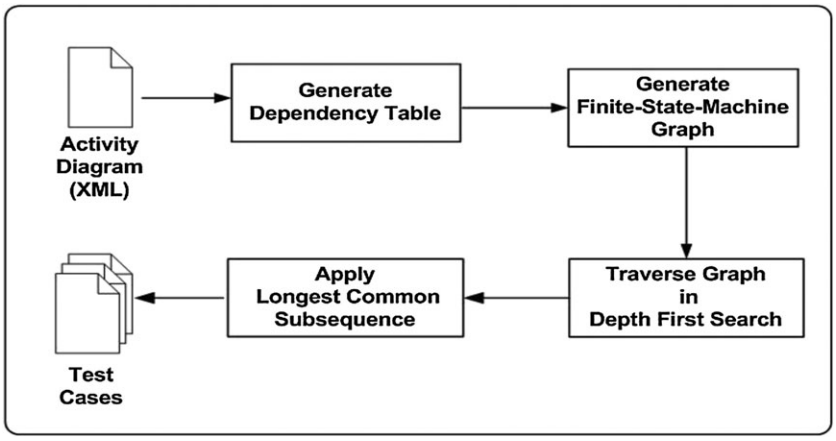

FIGURE 5 Automated test case generation approach

The activity diagram for the Test Task Execution module is shown in Figure 6. The overall flow within Test Task Execution module goes as follow:

1. Test Task Execution module retrieves activity diagram of the AUT from the Test Tasks repository. Because Test Tasks repository stores test requests submitted by the users with their input data required to complete testing process. The input data include activity diagram, APK, and GUI configurations file of the AUT as mentioned in Section 3.1.
2. The activity diagram of the AUT is converted to XML-based activity diagram file using XML parser tool (eg, Visual Paradigm).
3. Test Case Generation Module generates dependency table to store all information extracted from the XML-based activity diagram file.
4. A FSM graph is generated based on information included in the dependency table as described in Section 3.1.
5. The FSM graph is traversed in depth first search to find all possible test paths.
6. Longest common subsequences algorithm (LCS) is applied on the extracted test paths to remove paths that are included as a sub path in another path. The obtained paths after applying LCS represent test cases.
7. The Test Task Execution module retrieves APK file and GUI configurations file of AUT from the Test Tasks repository. The APK file and GUI configurations file are required to automatically execute the generated test cases.
8. The Test Task Execution module submits APK file of the AUT to the assigned virtual nodes where test cases will be executed using Appium tool.
9. Test Task Execution module installs APK file on Android Emulators of the assigned virtual nodes.
10. The Test Task Execution module executes the generated test cases simultaneously on multiple virtual nodes using Appium tool. The GUI configuration file of AUT includes GUI layout information as: GUI widget id, event required to fire GUI widget and widget type (eg, edit text or button). The GUI configuration file of AUT allows access to User Interface Elements (UIElements) of AUT.
11. The Test Task Execution module stores test results in the Test Results repository.
12. Test Task Execution module executes test cases simultaneously on multiple virtual nodes.
13. When Test Task Execution module finishes execution of all test cases, test report is generated to include results of the executed test cases. The test report is submitted to the user through the user interface.

4 | EXPERIMENTAL RESULTS

Experiments are performed to evaluate the feasibility of the proposed system. Four virtual machines are used to simulate a cloud-based environment and to allow the simultaneous processing of the test requests. The 4 virtual machines used in the experiments are created on 1 physical machine. VMware[54] is a software that allows users to create a cloud environment through using virtualization technique. VMware workstation[55] is the industry standard that allow users to create multiple VMs on 1 physical machine. MS SQL Server[56] is used to create the following 4 repositories: Test Tasks repository, Test Cases repository, Test Results, and VM Pool. Appium is installed on the VMs. The AUT APK file will be installed on Android Emulators before starting test cases execution.

A. Experiment 1:

The purpose of this experiment is to evaluate the proposed test case generation approach in terms of action coverage and activity coverage. The action coverage is defined as the ratio between the tested actions and all actions in the activity diagram. The activity coverage is defined as the ratio between the tested activities and all activities in the activity diagram. The proposed approach is compared with the functional traversal approach[29] and MonkeyRunner.[57] Functional traversal approach is a mobile application test method used in a TaaS platform. Functional traversal assigns weight to each operation. Then, it traverses operations with weight more than certain threshold value. MonkeyRunner is based on random events to test the AUT. In this experiment, 2 applications are selected as an applications under test: Simple Notepad application[58] and Clock application.[59]

Simple Notepad application is a native android application. It allows users to take notes easily. It has many features as create new checklist, sort, search, and more. Simple Notepad application consists of 6 GUI views. View 1 represents the home view where user can select to add new note, add new checklist, adjust application settings, add new folder, search, or sort saved notes. View 2 allows user to create new note by writing its title and body. View 3 presents new checklist view. View 4 allows user to add new folder. View 5 displays search or sort results. Finally, view 6 allows user to adjust application settings. Figure A.1 in the Appendix section shows the workflow of Simple Notepad application using UML activity diagram. The activity diagram is used to model the GUI-oriented user behavior for the Simple Notepad application in order to generate test cases automatically. Each GUI view is modeled as an activity in the shown activity diagram. Each action in activity diagram corresponds to event (s) exercised on the GUI controls.

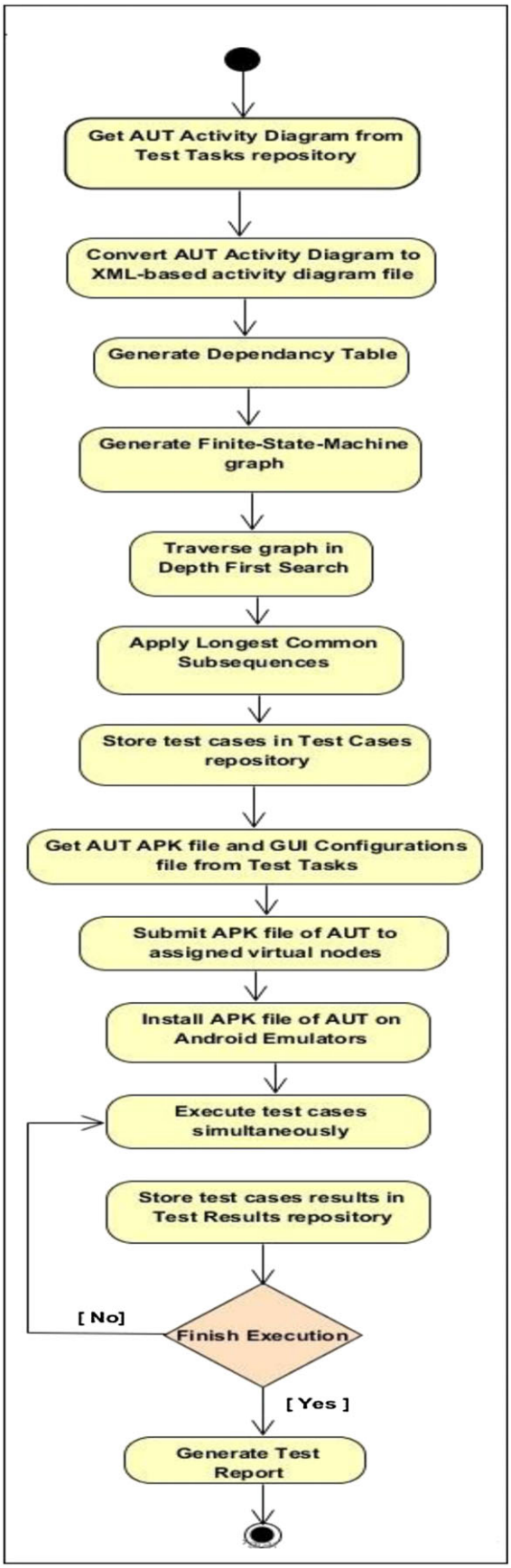

FIGURE 6 Test task execution module activity diagram

Activity diagram, AUT configuration file, and Simple Notepad.apk file are all submitted to the TaaS platform through user interface layer. First, test cases are automatically generated according to the previously explained test case generation approach. Number of generated test cases is 85 test cases; each test case represents a GUI user traversal behavior. The generated test cases cover all AUT activities (ie, GUI views) and actions. The proposed test case generation approach loops on each item in the menu widget. Therefore, each GUI control and each item in any menu widget in AUT is exercised at least once. As shown in Table 2, the proposed approach has higher activity and action coverage compared with the other 2 methods.

Similarly, Clock application is used as an AUT to verify the proposed test case generation approach. The Clock application includes many functionalities as set alarms, add timers, run stopwatch, and keep track of time around the world using World Clock. Figure A.2 in the appendix section shows the workflow of Clock application using UML activity diagram. Clock application consists of 7 GUI views. View 1 represents the home view where user can select to add new alarm, add world clock, start stopwatch, add timer, delete alarm, or show clock application information. View 2 allows user to create new alarm by setting the alarm hour/minutes/seconds. View 3 allows user to select city then add clock of the selected city. View 4 allows users to start or stop stopwatch. View 5 allows user to add new timer by defining the timer hour/minutes/seconds. View 6 allows user to delete certain alarm. Finally, view 7 displays information about the clock application. Each GUI view corresponds to activity in the activity diagram. The proposed test cases generation approach automatically generates 9 test cases. As shown in Table 3, the proposed approach guarantees higher activity coverage and action coverage compared with the other approaches.

B. Experiment 2:

The purpose of this experiment is to evaluate the effect of utilizing cloud environment resources for simultaneous test cases execution when many requests are sent to the TaaS platform vs the sequential execution. The evaluation is done in terms of the overall completion time required to execute all the submitted requests. The proposed system is compared with ADAutomation framework.[36] ADAutomation framework generates test cases based on UML activity diagram. It executes test cases sequentially on 1 machine. While the proposed system executes test requests simultaneously on multiple VMs. Figure 7 shows that the proposed system has lower overall completion time compared with the ADAutomation framework specially when increasing number of test requests, where the test requests enter both systems from the beginning. That is why both lines appear linear and are started from the origin. The proposed system has lower completion time due to the following reasons: (1) the scalability provided by the cloud based environment that allows simultaneous test execution on multiple VMs, (2) the tasks prioritization approach applied by

TABLE 2 Comparison of test case generation approaches for simple notepad application

Method	Feature	Activity Coverage	Action Coverage
MonkeyRunner	Random events	42.5%	36.7%
Functional traversal	Traverse operations based on weight	90.3%	62.2%
The proposed approach	Traverse all operations and activities	100%	100%

TABLE 3 Comparison of test case generation approaches for clock application

Method	Activity Coverage	Action Coverage
MonkeyRunner	41.8	36.2
Functional traversal	88.9%	61.5%
The proposed approach	100%	100%

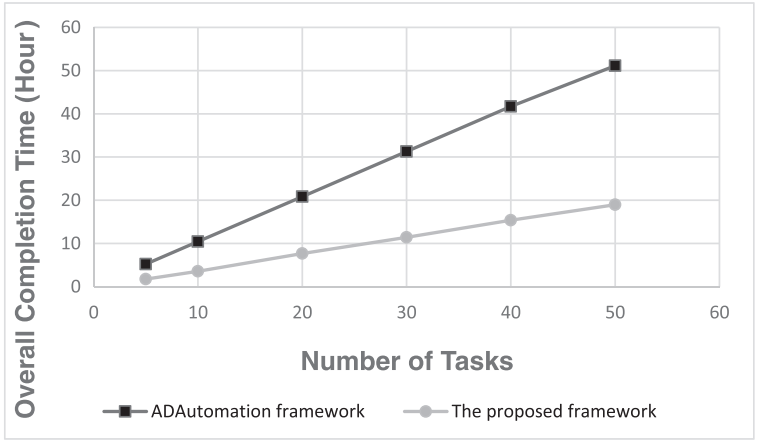

FIGURE 7 Overall completion time comparison for several test request

the scheduler module that guarantees satisfying test request deadline and reduces tasks waiting time, (3) the dynamically generated allocation plan executed by the resource allocator module that maximizes the resources utilization and guarantees system load balance, and (4) the test case generation approach used in the proposed system that reduces the number of automatically generated test cases by applying LCS algorithm, as well as guaranteeing high test coverage for AUT.

5 | CONCLUSION AND FUTURE WORK

TaaS offers opportunity to provide testing as a service through leveraging cloud resources and infrastructure. This leads to significant reduction in cost, time, and effort required for testing. Due to the widespread of mobile applications, necessity to test mobile applications increases. GUI testing gains much attention in order to ensure the reliability and the functionality of the mobile applications before releasing applications for the public use. Therefore, a system for mobile application GUI testing based on TaaS architecture is introduced in this paper, where all testing procedures are performed in a fully automated manner. The proposed system guarantees the automatic generation of test cases with higher coverage compared with other approaches. Moreover, a cloud-based resource load balance is achieved through the scheduler and the resource allocation module. The resource allocator module executes a dynamically generated allocation plan that maximizes the resources utilization. The scheduler module prioritizes tasks to guarantee satisfying test request deadline and reduce tasks waiting time. Experimental results show that the proposed system attains high test execution efficiency due to the simultaneous execution of test cases on multiple virtual nodes. Besides, the proposed system guarantees high test coverage compared with other approaches. In the future work, the proposed system can be extended to include many directions as (1) including more types of mobile applications testing as security testing and regression testing, (2) allowing GUI testing for different types of mobile applications running on different platforms, and (3) using real cloud environment like Windows Azure, which provides more scalability to the proposed system through renting more number of VMs.

ORCID

Amira Ali http://orcid.org/0000-0001-6932-7712

REFERENCES

1. Gao J, Bai X, Tsai W-T, Uehara T. Mobile application testing: a tutorial. *Computer*. 2014;47(2):46-55.
2. https://www.alliedmarketresearch.com/mobile-application-market
3. Kochhar PS, Thung F, Nagappan N, Zimmermann T, Lo D. Understanding the test automation culture of app developers. in software testing, verification and validation (ICST), 2015 IEEE 8th international conference on, pp. 1-10. IEEE, 2015.
4. Flora HK, Wang X, Chande SV. An investigation into mobile application development processes: Challenges and best practices. *Int J Mod Educ Comput. Sci*. 2014;6(6):1.
5. Nagappan M, Shihab E. Future trends in software engineering research for mobile apps. in software analysis, evolution, and reengineering (SANER), 2016 IEEE 23rd international conference on, vol. 5, pp. 21-32. IEEE, 2016.
6. Kirubakaran B, Karthikeyani V. Mobile application testing—challenges and solution approach through automation. in *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on*, pp. 79-84. IEEE, 2013.
7. Linares-Vásquez M, Moran K, Poshyvanyk D. Continuous, evolutionary and large-scale: a new perspective for automated mobile app testing. in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pp. 399-410. IEEE, 2017.
8. Mell P, Grance T. The NIST definition of cloud Comput Secur (2011).
9. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*. 2009;25(6):599-616.
10. Parveen T, Tilley S. When to migrate software testing to the cloud?. in software testing, verification, and validation workshops (ICSTW), 2010 third international conference on, pp. 424-427. IEEE, 2010.
11. Tilley S, Parveen T. SMART-T: migrating testing to the cloud. In *Software Testing in the Cloud*, pp. 19-35. Springer Berlin Heidelberg, 2012.
12. Zhenlong P, Zhonghui OY, Youlan H. The application and development of software testing in cloud computing environment. In Computer Science & Service System (CSSS), 2012 International Conference on, pp. 450-454. IEEE, 2012.
13. Priyadharshini V, Malathi A. Survey on software testing techniques in cloud computing. CoRR, abs/1402.1925, 2014.
14. Riungu LM, Taipale O, Smolander K. Research issues for software testing in the cloud. in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 557-564. IEEE, 2010.
15. Katherine AV, Alagarsamy K. Software testing in cloud platform: a survey. *International Journal of Computer Applications*. 2012;46(6):21-25.
16. Harikrishna P, Amuthan A. A survey of testing as a service in cloud computing. in computer communication and informatics (ICCCI), 2016 international conference on, pp. 1-5. IEEE, 2016.
17. Gao J, Bai X, Tsai W-T, Uehara T. Testing as a service (TaaS) on clouds. in service oriented system engineering (SOSE), 2013 IEEE 7th international symposium on, pp. 212-223. IEEE, 2013.
18. Wang W, Zhang X, Chen T, Wu X, Li X. Cloud computing based software testing framework design and implementation. Advances in Computer, Communication, Control and Automation 2012: 785-790.
19. Amalfitano D, Fasolino AR, Tramontana P. A gui crawling-based technique for android mobile application testing. in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pp. 252-261. IEEE, 2011.

20. Berenbach B, Paulish D, Kazmeier J, Rudorfer A. Software and systems requirements engineering: in practice. McGraw-Hill, Inc., 2009.
21. Wasnik C, Lingam C. Software Testing and Software Development Lifecycles. *Int J Comput Distrib Sys*. 2013;2(3).
22. Rumbaugh J, Jacobson I, Booch G. *The Unified Modelling Language User Guide*. Reading MA: Addison-Weslley; 1999.
23. Chen M, Mishra P, Kalita D. Coverage-driven automatic test generation for UML activity diagrams. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pp. 139-142. ACM, 2008.
24. Surpassed, Gartner Says Annual Smartphone Sales. Sales of feature phones for the first time in 2013. 2014.
25. Murugesan L, Balasubramanian P. Cloud based mobile application testing. in computer and information science (ICIS), 2014 IEEE/ACIS 13th international conference on, pp. 287-289. IEEE, 2014.
26. Gao J, Tsai W-T, Paul R, Bai X, Uehara T. Mobile testing-as-a-service (MTaaS)—infrastructures, issues, solutions and needs. in high assurance systems engineering (HASE), 2014 IEEE 15th international symposium on, pp. 158-167. IEEE, 2014.
27. Kaur K, Kaur A. Cloud era in mobile application testing. in computing for sustainable global development (INDIACom), 2016 3rd international conference on, pp. 1057-1060. IEEE, 2016.
28. Tao C, Gao J, Li B. Cloud-based infrastructure for mobile testing as a service. In Advanced Cloud and Big Data, 2015 Third International Conference on, pp. 133-140. IEEE, 2015.
29. Zhang S, Pi B. Mobile functional test on TaaS environment. in service-oriented system engineering (SOSE), 2015 IEEE symposium on, pp. 315-320. IEEE, 2015.
30. Mahmood R, Esfahani N, Kacem T, Mirzaei N, Malek S, Stavrou A. A whitebox approach for automated security testing of android applications on the cloud. in automation of software test (AST), 2012 7th international workshop on, pp. 22-28. IEEE, 2012.
31. Zhang T, Gao J, Cheng J, Uehara T. Compatibility testing service for mobile applications. in service-oriented system engineering (SOSE), 2015 IEEE symposium on, pp. 179-186. IEEE, 2015.
32. Grønli T-M, Ghinea G. Meeting quality standards for mobile application development in businesses: a framework for cross-platform testing. in system sciences (HICSS), 2016 49th Hawaii International Conference on, pp. 5711-5720. *IEEE*, 2016.
33. Starov O, Vilkomir S. Integrated TaaS platform for mobile development: architecture solutions. in automation of software test (AST), 2013 8th international workshop on, pp. 1-7. IEEE, 2013.
34. Versluis G. Creating and running tests with xamarin test cloud. In *Xamarin Continuous Integration and Delivery*. Berkeley, CA: Apress; 2017:71-91.
35. http://tools.perfectomobile.com/
36. Li A, Qin Z, Chen M, Liu J. ADAutomation: an activity diagram based automated GUI testing framework for smartphone applications. in software security and reliability, 2014 eighth international conference on, pp. 68-77. IEEE, 2014.
37. Anbunathan R, Basu A. Automatic test generation from UML sequence diagrams for android mobiles. *International Journal of Applied Engineering Research*. 2016;11(7):4961-4979.
38. Ali A, Badr N. Performance testing as a service for web applications. in intelligent computing and information systems (ICICIS), 2015 IEEE seventh international conference on, pp. 356-361. IEEE, 2015.
39. Zheng Y, Cai L, Huang S, Wang Z. VM scheduling strategies based on artificial intelligence in cloud testing. in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*, pp. 1-7. IEEE, 2014.
40. Liu C-H, Chen S-L. Evaluation of cloud testing strategies based on task decomposition and allocation for improving test efficiency. in *Applied System Innovation (ICASI), 2016 International Conference on*, pp. 1-4. IEEE, 2016.
41. Paradigm, Visual. Visual paradigm for UML-UML tool for software application Development 2013.
42. Janicki M, Katara M, Pääkkönen T. Obstacles and opportunities in deploying model-based GUI testing of mobile software: a survey. *Software Testing, Verification and Reliability*. 2012;22(5):313-341.
43. Cartaxo EG, Neto FGO, Machado PDL. Test case generation by means of UML sequence diagrams and labeled transition systems. In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pp. 1292-1297. IEEE, 2007.
44. Berenbach B, Paulish D, Kazmeier J, Rudorfer A. Software & systems requirements engineering: in practice. McGraw-Hill. In: *Inc.* ; 2009.
45. Gudmundsson V, Lindvall M, Aceto L, Bergthorsson J, Ganesan D. Model-based testing of mobile systems--an empirical study on QuizUp android app. arXiv Preprint arXiv:1606.00503 2016.
46. Sipser M. *Introduction to the Theory of Computation. Vol. 2*. Boston: Thomson course Dent Tech; 2006.
47. Heineman GT, Pollice G, Selkow S. Algorithms in a nutshell: a practical guide. O'Reilly Media, Inc., 2016.
48. Hirschberg DS. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*. 1977;24(4):664-675.
49. APPIUM, URL http://appium.io/index.html? lang=en
50. Singh S, Gadgil R, Chudgor A. Automated testing of mobile applications using scripting technique: a study on Appium. *International Journal of Current Engineering and Technology (IJCET)*. 2014;4(5):3627-3630.
51. APPIUM, URL http://appium.io/index.html?lang=en
52. http://robotium.com
53. http://calaba.sh/
54. Rosenblum M. Vmwares virtual platform. In *Proceedings of hot chips*, vol. 1999, pp. 185-196; 1999.
55. https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html
56. Agrawal S, Chaudhuri S, Kollar L, Marathe A, Narasayya V, Syamala M. Database tuning advisor for microsoft sql server 2005. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 930-932. ACM, 2005.
57. MonkeyRunner, http://developer.android.com/tools/help/monkeyrunner_concepts.html

58. https://apkpure.com/simple-notepad/org.mightyfrog.android.simplenotepad
59. https://apkpure.com/clock/com.google.android.deskclock

How to cite this article: Ali A, Maghawry HA, Badr N. Automated parallel GUI testing as a service for mobile applications. *J Softw Evol Proc*. 2018;30:e1963. https://doi.org/10.1002/smr.1963

APPENDIX A.

APPLICATIONS UNDER TEST ACTIVITY DIAGRAMS

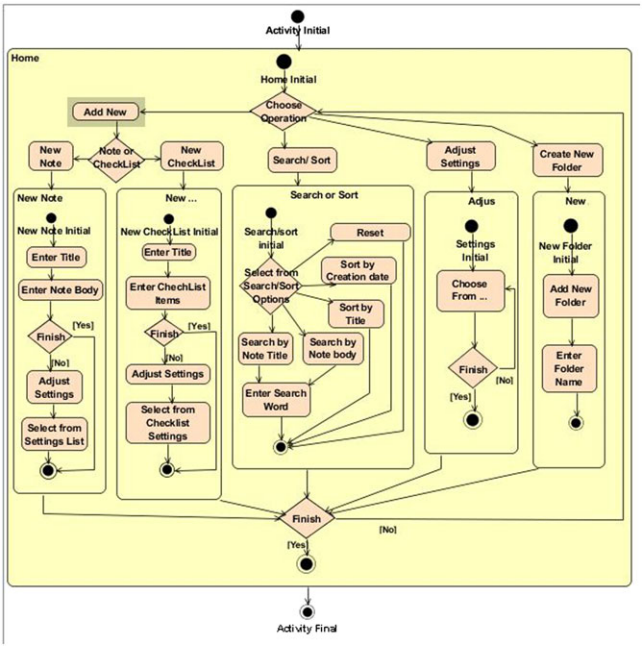

FIGURE A.1 Simple notepad application activity diagram

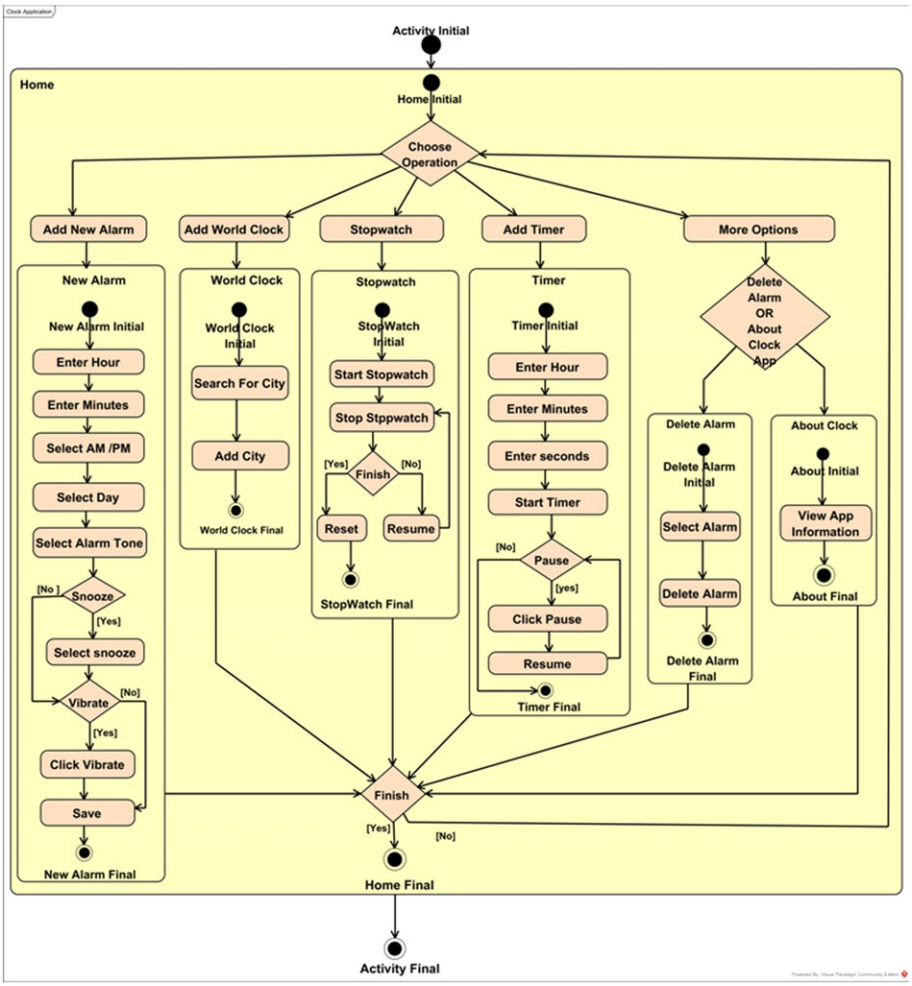

FIGURE A.2 Clock application activity diagram