

Cloud-Based Mobile App Testing Framework: Architecture, Implementation and Execution

Isabel Karina Villanes

Institute of Computing
Federal University of Amazonas
Manaus – AM, Brazil
isabelkarina@icomp.ufam.edu.br

Silvia Meireles

Institute of Computing
Federal University of Amazonas
Manaus – AM, Brazil
silvia@icomp.ufam.edu.br

Arilo Claudio Dias-Neto

Institute of Computing
Federal University of Amazonas
Manaus – AM, Brazil
arilo@icomp.ufam.edu.br

ABSTRACT

The growth in the use of mobile devices is notorious due to the multiple functionalities they offer. The time between the release of new device models and mobile platform updates is very short, and this has a direct influence on the quality of mobile applications, because these applications need to be compatible with new mobile devices and the different versions of mobile platforms. As a negative consequence, the quality of mobile apps would be lower than expected. Therefore, to ensure mobile application quality, many services for mobile test are offered as Cloud Testing. Thus, this work proposes a Mobile Cloud Testing Framework, called AM-TaaS, that meets these needs. AM-TaaS facilitates the test environment setup and configuration and covers a range of mobile devices and platforms. We also describe the architecture and implementation of the proposed framework. With the use of AM-TaaS framework, it is possible to perform mobile app testing on different mobile emulators/devices.

Keywords

Mobile testing, automated testing, mobile cloud testing

1. INTRODUCTION

There are hundreds of different mobile devices produced by several vendors with different software and hardware features in the market. During the last decade, mobile devices have evolved into powerful, dynamic, connected and smart devices able to deliver all kinds of services at our fingertips and mobile applications¹ (or just mobile apps) are distributed with them. Currently, mobile apps demand is higher than ever. While mobile apps were originally offered for basic tasks, such as weather information, games (e.g.: *the Snake game*), email, nowadays they are more sophisticated with features like GPS mapping and location-based services, banking, video streaming and mobile medical apps.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SAST, September 19 - 20, 2016, Maringa, Parana, Brazil

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-4766-2/16/09...\$15.00

DOI: <http://dx.doi.org/10.1145/2993288.2993301>

Technological advances and the use of mobile devices along with the wide variety of models available in the market resulted in demand for new and more mobile apps, in order to facilitate the user's daily tasks. Recent research published by Statista [1] states that the number of downloads of mobile apps is expected to reach 268.69 billion in 2017. This number is expected to grow in the future and mobile developers have to meet the demand for quality in order to achieve highlight on mobile app stores. The use of mobile apps also generated revenues in 2015, which amounted to 41.1 billion dollars. By 2020, it is projected that users will spend more than 101 billion dollars in mobile apps through app stores [2].

Mobile app development is time consuming, expensive and difficult because there are hundreds of different mobile devices, with different operating systems (OS), browsers, screen sizes, video resolution, various natives APIs, software features, hardware components, among other attributes [3][4][5][6]. In addition, as the complexity of an application increases, the complexity of the test also will equally increase due to the use of new test cases needed as well as new tools. This scenario represents a challenge for software engineering applied to mobile platforms. Moreover, the diversity of mobile devices and platforms is pointed out as the main problem related to the development and testing of mobile apps [7]. The frequent upgrades of mobile devices and technologies even exacerbate this issue.

This work is related to software testing of mobile apps. With the growth in the use of smartphones, mobile app testing is also experiencing growth and more sophisticated as compared to before. It has been used for reducing faults and enhance users' confidence in the mobile app. Although the benefits of performing mobile app testing are known, the costs of this activity are high. Therefore, a strategy to reduce these costs is necessary. According to Muccini *et al.* [3], automation is among the most important means for both keeping the cost of testing low and improving quality of mobile apps. To achieve this objective, a testing environment that uses a vast diversity of mobile devices/platforms to evaluate a mobile app for which it was developed is necessary.

Mobile applications are usually developed by small teams (one or two people) responsible for conception, design, and development [8] as well as testing the mobile app to ensure quality. Consequently, it is normal that the team uses local infrastructure (e.g. personal computers and mobile devices) as test environment. This represents a challenge for mobile app testing. Real and emulated devices are used for testing. Emulation-based and device-based testing are mobile app testing approaches, and they have pros and cons [3]. The emulation-based approach involves using an emulator that comes along a mobile platform's software development kit. It allows the development and testing of mobile apps without the use of a real device [9]. Using emulated devices, a tester can easily switch to

¹ Software applications developed for mobile platforms [6]

different device types by simply loading the appropriate device profile. They could be used to assess system functionality in limited contexts. Some challenges in such an approach are to simulate device-level variables, such as mobile network, Wi-Fi capabilities, data test location and validation of a set of gesture functions [10]. On the other hand, device-based testing can verify device-based functions and different behaviors. It requires setting up a testing laboratory, and purchase real mobile devices, which represent a high testing cost. Using real devices, we would be able to validate their underlying mobile networks via reconfigurations and selections in a test environment. This approach presents some challenges such as coping with rapid changes in mobile devices/platforms and large-scale tests that require many mobile devices [10].

In order to realize mobile testing, developers need to spend time studying mobile testing tools, checking if these tools fit their needs and writing test scripts. This learning process takes a while and it could be a challenge for mobile testing, since time to market is a very important aspect to the success of a mobile app. Tao and Gao [11] state that an effective test automation for mobile applications must include solutions that are compatible, deployable, and executable on different mobile platforms, devices, network, and appliance APIs.

Recent studies have proposed cloud-based testing services such as Perfecto Mobile², SauceLabs³, SOASTA⁴, TestDroid⁵, Amazon Web Services AWS Device Farm⁶ and Firebase⁷ as alternatives to provide test environments. Cloud-based testing service is a new model to provide testing capabilities as a service to end users using a cloud infrastructure [12]. When compared to other approaches, this one can be more cost-effective for testing large-scale applications, and it is much more effective for supporting diverse testing activities on mobile devices [10]. Thus, the use of cloud computing resources for software testing could help and improve this issue. Testing as a Service (TaaS) is becoming a hot research topic that is being used on different software platforms for Regression Test [13], Web Security [14], Unit Testing [12][15] and Load Web Testing [16]. However, testing in mobile apps faces challenges, characteristics inherent to cloud computing and mobile testing. For instance, the following issues are identified in [17]: lack of standards, mobiles test environments, well-defined test models and coverage criteria to address distinct needs in mobile testing.

In order to facilitate and improve these issues, this paper presents a Cloud-Based Mobile Testing Framework called *Automated Mobile Testing as a Service* (AM-TaaS) which aims at reducing the effort to execute mobile testing applications. This paper describes the architecture, implementation and execution of the proposed framework. The main idea is to create a cloud-based service that allows running testing on a variety of emulated/real mobile devices, contributing to improve the quality of mobile apps.

This work explains the design and implementation of the AM-TaaS framework to support mobile app testing for addressing challenges by testers. The remainder of this paper is organized as follows. Section 2 provides the background and a summary regarding testing as a service, mobile testing, and related frameworks and tools for mobile cloud testing. Section 3 outlines the architecture and Implementation of AM-TaaS. While the AM-TaaS execution is described in Section 4. Finally, conclusion and future work are summarized in Section 5.

2. BACKGROUND

2.1 Testing as a Service (TaaS)

During the last years, a new service model, known as *Testing as a Service* (TaaS) or *Cloud-based Testing*, has gained enormous popularity due to its scalability. TaaS is a hot topic issue at the moment [12][18]. There is no detailed definition about what is TaaS. According to [19], TaaS is automated software testing as a cloud-based service. In [12], the authors define TaaS as a new model to provide testing capabilities to end users. Moreover, according to [20], TaaS promotes a Cloud-based testing architecture to provide online testing services following a pay-per-use business model.

Inki *et al.* [18] indicate the cloud-based environment as the most recommended environment to migrate and to perform software testing, due to the possibility of using cloud resources to perform large-scale tests, since all test activities and management are brought to the cloud [21]. TaaS combines two ideas: (1) offering software testing as a competitive and easily accessible web service, and (2) doing fully automated testing in the cloud, to harness vast, elastic resources toward making automated testing practical for real software [19].

The cloud computing is divided into three types of layers: IaaS, PaaS and SaaS [22].

- **Infrastructure as a Service (IaaS):** clients buy the infrastructure, such as servers and storage, and the model is pay-as-you-go on demand.
- **Platform as a Service (PaaS):** it provides a platform in which the software can be developed and deployed allowing the customer to manage, run and develop the application.
- **Software as a Service (SaaS):** it is deployed over the Internet, managed by a third-party vendor and its interface can be accessed on the client's side.

TaaS is executed as part of these three layers. It can be used to test the infrastructure, platform and software, or all combined, as shown in Figure 1.

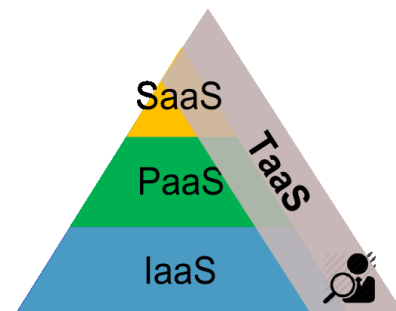


Figure 1. TaaS and its effect on the three different cloud layers.

According to [23], TaaS is important for the following reasons:

- Cost reduction by leveraging computing resources in clouds.
- Scalable test environments with virtualization.
- On-demand testing service in 365/7/24.
- Model Pay-as-you-test.
- Quality certification by third parties.

² <http://www.perfectomobile.com>

³ <https://saucelabs.com/features>

⁴ <http://www.soasta.com/solutions/cloud-testing>

⁵ <http://testdroid.com>

⁶ <https://aws.amazon.com/device-farm/>

⁷ <https://firebase.google.com/docs/test-lab>

2.2 Mobile Testing

The term Mobile Testing refers to “testing activities for native and web applications on mobile devices using well-defined software testing methods and tools to ensure quality in functions, behaviors, performance, and quality of service, as well as features, such as mobility, usability, interoperability, connectivity, security, and privacy” [10]. Non-functional testing focuses in requirements such as performance, usability, and security while functional testing focuses in basic functionality such as service functions, mobile Web APIs, external system behaviors, system-based intelligence, and user interfaces (UIs) [24].

Setting up a mobile test environment for multiple apps on each mobile platform for a range of devices is tedious, time-consuming, and expensive [10]. Different mobile test infrastructures are defined, such as emulation-based, device-based, crowd-based testing and cloud testing. According to [25], mobile app testing has some similarities to website testing as both involve validation in many environments (smartphones and browsers, respectively). However, mobile testing has specific testing characteristics like different platforms, operating systems, screen sizes, memory and processing capacity, models by vendors and communication by different networks. Testing all combination from these characteristics could be unfeasible. Thus, it is necessary to perform compatibility testing for discovering failures of the application caused by the variety in hardware and configuration profiles and failures related to use the application on non-targeted devices [24]. This paper focuses on functional testing of Android mobile applications.

Concerning the use of mobile testing open source tools for functional testing, they offer different kinds of tests. Table 1 presents a list of functional testing, open source tools for mobile apps and some important attributes used by cloud-based testing services.

2.3 Mobile Testing as a Service

The growth in the use of mobile apps in the last years raised the idea of *Mobile Testing as a Service* (Mobile TaaS). Mobile TaaS provides on-demand testing services for mobile apps and/or SaaS to support software validation and quality engineering processes by leveraging a cloud-based scalable mobile testing environment to assure pre-defined given Quality of Service (QoS) requirements and service-level-agreements (SLAs) [17]. This service is based on Mobile Cloud Computing (MCC), which is the intersection between both mobile and cloud computing [26], as illustrated by Figure 2.

Mobile TaaS uses the cloud infrastructure to offer a “pay-as-you-test” model to ensure resource sharing and cost reduction. Nevertheless, in [27] the authors claim in mobile app testing there are various unique factors that challenge the testing process, such as device diversity (screen size, screen orientation, free memory, chipset, architecture), operating system, network and runtime environment. The use of mobile devices and apps is evolving exponentially. A research published in [28] explains that cloud services will evolve by 17.3% until 2018 and therefore more attention about test mobile apps is required.

Compared to conventional mobile testing, Mobile TaaS is composed of three main environments: Emulation-Based Mobile Testing on Cloud, Simulation-Based Mobile Testing on Cloud and Device-Based Mobile Testing on Cloud [17], as shown on Figure 3.

Many cloud-based frameworks are available for mobile app testing for various purposes [29]. Each framework focuses on a different type of test. This section briefly summarizes the related research on

frameworks and tools in mobile app testing using a cloud-based infrastructure.

Table 1. Attributes of Mobile Testing Open Source Tools.

Attributes	Appium	Monkeyrunner	MonkeyTalk	Calabash	Robotium
Emulation-Based Test	X	X	X		X
Device-Based Test	X	X			X
Simulator-Based Test			X		
Testing For Native Apps	X	X	X		X
Testing For Web Apps	X		X		
GUI-Based Testing	X	X	X		
Accepting Test				X	
Supported Script Language	X	X	X	X	X
Support any Language	X				

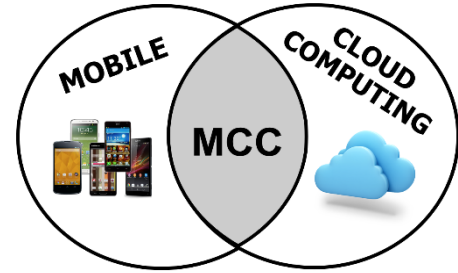


Figure 2. Mobile Cloud Computing –MCC

Mahmood [30] presents *EvoDroid*, a tool that aims to find a set of test cases that maximize code coverage. From the source code, *EvoDroid* extracts two types of models, representing the app’s external interfaces and internal behaviors, in order to generate test cases automatically. Tests are performed in parallel on Android emulators deployed on the cloud. The cloud environment used was a virtual server instance on Amazon Elastic Compute Cloud (Amazon EC2).

Oliveira and Duarte [31] present a cloud-based testing framework that focuses on parallelizing tests execution of a unit tests. JUnit⁸ was used as the main tool to create tests. These tests were distributed by many virtual machines on the cloud. They compared the time execution between a local environment and an instance on Amazon Web Services (AWS) that was used as a virtual server.

The *Mobile Testing as a Service* (MTAAS) framework was proposed in [32] using a virtual machine created on SOASTA⁹ company, on which real mobile devices were connected to the virtual machine through wireless network. Load testing performance was measured in terms of response time; some actions were recorded and tested. Three devices, three open source mobile apps and five virtual users were simulated.

For functional testing, Starov *et al.* [33] present *Cloud Testing of Mobile Systems* – CTOMS, a framework focused on Testing as a Service for mobile development. This framework comprises of two components: The *Master Layer* and *Client Layer*.

⁸ <http://junit.org>

⁹ <http://www.soasta.com>

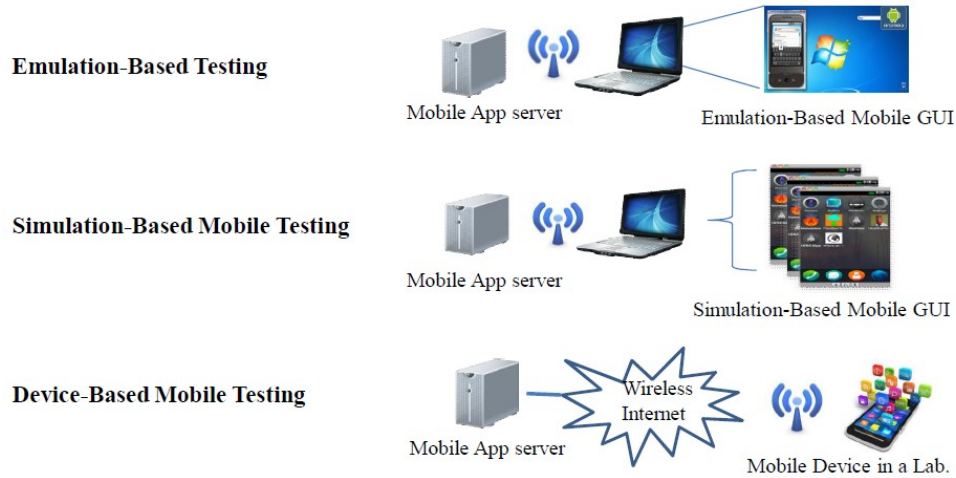


Figure 3. Three Cloud-Based Mobile Test Environments.

The process of test execution is performed on real devices connected to the *Master Layer* component. The pairwise testing combinatorial approach is used as a way to cover different combinations of hardware and software parameters.

Firebase Test Lab provides cloud-based infrastructure for testing Android apps. With one operation, it is possible to initiate testing on mobile apps across a wide variety of devices (*13 devices for free edition*) and device configurations. Test results—including logs, videos, and screenshots—are made available. Firebase Test Lab is a subproject of Firebase, which is part of Google Developer, launched on May-2016. Firebase offers an analytic product named Firebase Analytics. It is a brand-new, free and analytic solution for mobile apps.

We also analyzed commercial cloud-based testing services related to platform, environment for testing and test approach. Some cloud-based services offer "Manual testing" that is related to the web testing service. For instance, in *Testdroid*, mock mobile devices are used. Table 2 shows the summary of cloud-based testing services analyzed. On the other hand, Table 3 shows the use of open source tools that are offered as a way of integration by cloud-based testing services.

All Cloud-Based Mobile Testing Services use real devices for testing; only *SauceLabs* offer emulators. Two companies use simulated mobile devices for test. This is performed by the web browser since the mobile device is accessible by web. Regarding the use of open source frameworks, many of them use the *UIAutomator* tool due to the ease of obtaining information about the element's id/name of an application.

In the next section, we explain our proposed Framework, called AM-TaaS, which aims to help developers and testers who intend to evaluate the quality of their mobile apps in various mobile devices in a short deadline.

3. AUTOMATED MOBILE TESTING AS A SERVICE (AM-TaaS FRAMEWORK)

Test automation has a huge impact on mobile app quality, ensuring that app will work according to the characteristics defined in the development process reducing human interference during the tests execution. In this scenario, obtaining mobile app quality has not been an easy task, since tests require the app execution in a large number of possible combinations of mobile devices, operating systems (SO) and

different characteristics about screen size, memory capacity, etc. The proposed framework aims to help developers with this task.

Table 2. Attributes of Cloud-Based Mobile Testing Services.

Attributes		SauceLabs	SOASTA	Perfecto mobile	TestDroid	AWS Device Farm	Xamarin
Platform	Android OS	X	X	X	X	X	X
	iOS	X	X	X	X	X	X
	Windows OS	X		X			
	Fire OS					X	
Environment	Emulation-based testing	X					
	Device-based testing	X	X	X	X	X	X
	Simulated-based testing				X		X
Type Test	Mobile web testing (Manual testing)	X	X		X		X
	Touch testing		X				
	Testing for native apps	X	X	X	X	X	X
	Testing for hybrid apps	X	X	X	X	X	X
	Record and replay				X		
	Mobile performance		X				

Table 3. Open Source Tools used by Commercial Tools.

Could-based Services x Mobile Test Tools	Appium	Calabash	Cucumber	Selenium	UIAutomator
Saucelabs	X				X
SOASTA				X	X
Perfecto mobile	X		X	X	
TestDroid	X				X
AWS Device Farm	X	X			X
Xamarin		X	X		

In this section, we will briefly summarize the design of various components of the Automated Mobile Testing as a Service (AM-TaaS) framework. It aims at offering mobile cloud testing service using emulation/device-based testing. AM-TaaS considers mobile cloud services for developers and mobile app testers. This framework offers automated testing that is based on test criteria for mobile apps published by AQuA [34], but other sources to provide tests can also be included, including test scripts provided by the tester.

AM-TaaS is based on the Architectural Model for Cloud Computing [35]. The overview of AM-TaaS is shown in the Figure 4. It comprises three distinct components namely: *Service Request*, *Infrastructure* and *Service*.

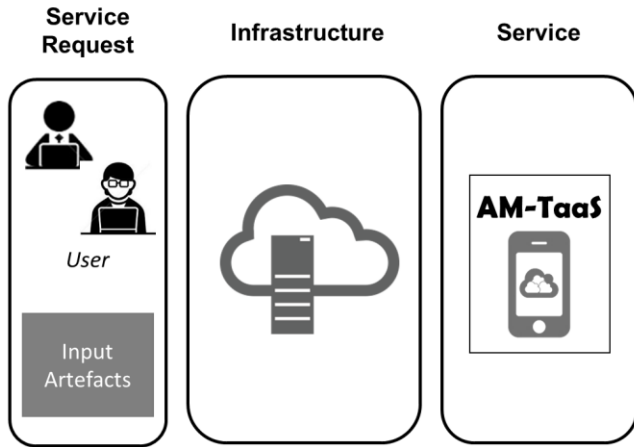


Figure 4. AM-TaaS overview.

The main objective of AM-TaaS is to help to reduce the effort to perform tests on multiple devices/emulators. To implement the proposed framework, we chose Android as an initial mobile platform target, because Android is the most popular and widespread mobile OS, according to several statistics agencies.

3.1 Choosing the Infrastructure

In order to meet the requirements of a scalable cloud infrastructure, with broad access to resources and services on demand, the implementation of AM-TaaS can be in different ways, which will depend on the resources that we have access to. The following list presents AM-TaaS's possible implementation:

1. *Local Server*: The service implementation can be allocated on a local server and accessed via a web interface. The server has its own resources for databases and data storage. Emulators are created or mobile devices are connected to the server.
2. *Infrastructure in the cloud*: This type of system includes all features and services on the cloud such as a virtual machine VM, emulators (instantiated on VMs), devices (connected to specific servers for management).

The second option could be the best choice, because it is entirely based on cloud resources. However, this is a cost to be paid for the cloud resources that have to be used. On the other hand, the first option for implementation on a local server is also feasible, as it enables the use of all server owned resources and allows access via a web interface that facilitates access to the service, but the performance level will be lower in relation to option two. As relates to economy, the first option is less expensive and more feasible to be implemented, which is the main goal of this paper, to present the service model for mobile testing.

This paper aims to present the test service model based on cloud resources. After the analysis of the two options for AM-TaaS' implementation, the first option was chosen (Local Server) that allowed us to implement and provide the service model with all its features, but without the performance of cloud resources.

3.2 Components of AM-TaaS framework

The AM-TaaS framework uses a layered design to reduce the complexity of the whole framework. The layers are *User Interface*, *Information* and *Resource*. Each layer is made up of specific components that work together to implement the entire test service process. The components and their relationship are shown in Figure 5 and Figure 6, respectively.

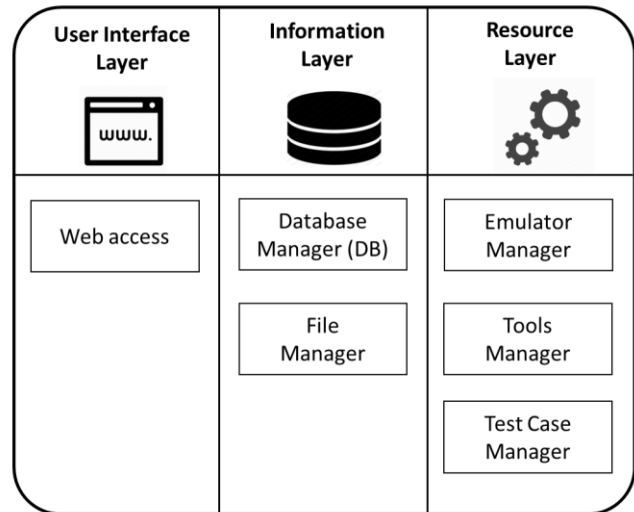


Figure 5. Components of AM-TaaS.

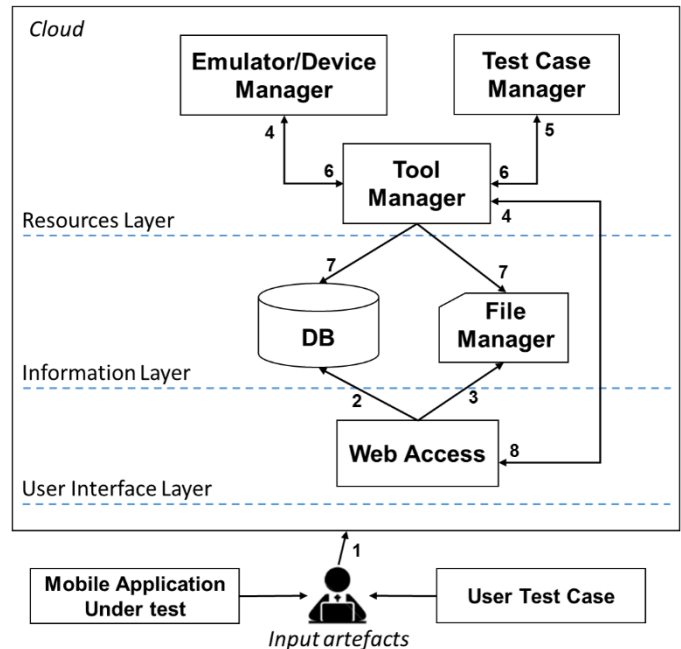


Figure 6. Relationship between the components of each Layer.

Each layer contains specific components that work together to perform the automated test, as described in the followings high level steps:

- 1) A logged user sends input artifacts to the AM-TaaS Framework.
- 2) The data of users and information are saved in the database.
- 3) A directory is created to the saved input artifacts.
- 4) Emulators are instantiated or devices are connected to be used.
- 5) The test scripts are called to run.
- 6) The test scripts are executed on emulators or devices.
- 7) The execution's results are saved in the database and in the File Manager.
- 8) Results are shown in the user interface.

3.2.1 User Interface Layer

a) Web Access: This component is intended to expose the service model logic, allowing interaction among users who perform the request of a service and the infrastructure in which the AM-TaaS framework is located. It has a GUI interface that allows easy access to this layer. The access to a service through this layer presents the following advantages:

- It is not necessary for the user to know the technology used behind this layer.
- The computational power requirement of the user is minimal, because the majority processing is performed on the server.

On this component, users can review information and characteristics about emulators and devices before starting with the test execution. Information presented are API version, Ram memory and screen size. Input devices that are needed to start execution test on AM-TaaS test service are sent through this layer. Input artefacts are binary files (APK) and Test Scripts. This component is graphically presented in Figure 7.

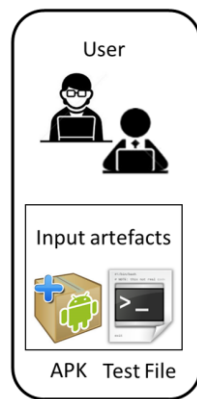


Figure 7. Input Artefacts of AM-TaaS.

3.2.2 Information Layer

Two types of storage represent this layer: Database and File Storage.

a) Database Manager:

This component is responsible for managing the data framework, user data, information about characteristics and status of emulators. User data is important because when a request for the service is performed, the File Manager and Emulators use user's data. When an emulator is used, its status is updated and other users cannot stop testing the service execution process.

b) File Manager:

The components that make use of this feature are: Web Access and Tools Manager. As the AM-TaaS works with physical files (binary file of mobile app, test scripts files set as defaults and script files that

users send, logs), this component is also responsible for creating a directory of folders each time a user requests the test service. The structure of the created directory is shown in Figure 8. Results will be erased when a new execution of the service is started, because of the limited storage of the local server. If the service will be offered in the cloud, all data would be available for future reference.

Contents on folders are as follow: The **AppFile** folder contains the binary file of application to be tested (apk). The folder **ScriptFiles** contains all user script files that will be executed on emulators. The **Devices** folder contains information about the emulators that the user has chosen. The folder **Images** contains screenshots of application test cases; the folder **Log** contains the script test's log and the folder **DeviceLog** contains each emulator's log by separated.

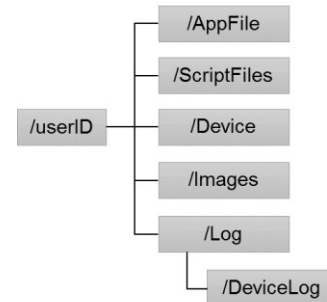


Figure 8. Directory structure of Files Manager component.

3.2.3 Resources Layer

This layer is composed of tools, which are responsible for managing and starting emulators and devices. When emulators or devices are ready for use, the automated test scripts are executed on each emulator/device selected by the user. It comprises three sub components, described below.

a) Emulator/Device Manager:

Considering that the current infrastructure defined for the proposed framework is based on the Android platform, this component uses the SDK Manager package (Software Development Kit) to manage the emulators/devices. This component is also responsible for restarting the emulators after use and clean all user data. Furthermore, it checks whether emulators/devices chosen by users are available for using by a new user.

This component uses the Android Virtual Devices – AVD Manager, which creates emulators with different characteristics and versions of the Android OS. In this work, these emulators will be called AVDs. An AVD mimics the behavior of a real mobile device. When the emulator is created, a file is generated called "AVD configuration" which describes how to behave to simulate a real device. The information that contains this file are:

- Version of the Android platform to emulate (API level).
- Screen size, density and screen resolution.
- SD card size
- RAM size for emulator.
- CPU type.

b) Test Case Manager:

The main objective of this component is to host and manage the testing criteria that were automated in this study (called default test cases) and the Test Scripts that are sent by users. AM-TaaS runs all tests (default and user Tests) for all applications that are sent to the

² <http://www.appqualityalliance.org>

Framework. Table 4 explains the test cases that are suggested by the AQUA testing criteria for mobile apps [34].

Table 4. AM-TaaS Default Automated Test Cases.

ID	TEST TITLE	TEST DESCRIPTION
1	OTA Install (a)	Install the App
2	OTA Install (b)	Install the App with no space left in device
3	OTA Uninstall	Uninstall the App

On the other hand, the test cases that are sent by users are also managed and executed. The structure is variable by virtue of the variety of mobile apps that can be tested using the AM-TaaS. The user is allowed to send one script test file or a set of script test cases. They are sent in compressed format in a file with extension "zip". To run, they will be decompressed and executed for all emulators that the user has selected.

c) Tools Manager:

This component comprises many tools. Each one has an important task in the whole testing process. They are:

- *Android Debug Bridge* (ADB) [36]: it is a command line tool that lets one work and communicate with emulators or connected devices. This tool allows the state control of a smartphone Android. Thus, for example, through ADB it is possible update the system, run shell commands, manage port forwarding or copying files. It was used to manage and control all the emulated devices on AM-TaaS.
- *Monkeyrunner* [37]: this tool comes along with the Android SDK framework. It is designed for testing apps at the functional level and for running unit test cases. To interact with the application this tool uses coordinates (x,y). This feature could be considered as a con because it makes scripts screen size dependent. The test scripts are written in the Python programming language.
- *AndroidViewClient* [38]: it is a test automation tool that enables the creation of Python test scripts. It is independent of *Monkeyrunner*. Some pros of this tool are that the interaction with the component views of an application is completely device independent and one does not have to take into account different screen sizes, resolutions or densities as the operations can be specified by view attributes instead of (x,y) coordinates. It supports devices with *UIAutomator* [39].
- *UIAutomator* (supported on Android API 18 or higher): it is part of the *Android SDK* that comes along with *UIAutomatorViewer* and lets one list all UI (User Interface) objects of a mobile app. It helps in the construction of reusable test scripts.

3.3 Implementation of AM-TaaS

To support the cloud testing service model and in order to present a demonstrative version of the Mobile Cloud Testing Service, in this section we present details of the implementation.

a) Environment:

Due to the chosen infrastructure for the implementation of AM-TaaS, the local server has the following features: 8GB of RAM, Intel Core i5 3.0 GHz, 100 GB of storage, and the operating system Linux Ubuntu 64-bit. The Android SDK Manager was installed and configured, we defined the local variables for "*ANDROID_SDK_HOME*", "*ANDROID_SDK_HOME/tools*" and "*ANDROID_SDK_HOME/platform-tools*", and they are responsible

for the correct functioning of the SDK and tools that AM-TaaS uses, such as ADB, MonkeyRunner. On the other hand, it was required to install some libraries (*ia32-libs*) to support 32-bit applications.

To create emulators, the Android Device Manager (AVD Manager) was used. Also it was necessary to specify the characteristics of emulators that would be used. It is important to note that an analysis of the most used Android versions and the most common screen sizes was made. This information was obtained from the Android site¹⁰.

Apache is used as the web server that allows user interaction with the AM-TaaS infrastructure. In order to meet the AM-TaaS needs, it was necessary to set up some features like the size limit for files to be uploaded to the framework, as well as the maximum size of such files.

The size of mobile apps is an important characteristic for the framework, because these binary files are sent to the framework via http. Thus, it was necessary to increase the maximum file sizes that would be uploaded to the server.

b) Web Access (Front end):

In order to follow a web development standard, the User Interface has been implemented based on MTControl tool [40]. This tool was implemented using Yii Framework that allows one to create web interfaces based on the MVC (Model View Control), PHP was the language used.

c) Database and File Manager:

The Database Manager component is composed of a MySQL database. In contrast, the files (binaries, logs, test scripts) were stored directly on the hard drive organized in directories as described in File Manager component.

d) Test Case Manager:

Its component is composed of the default test cases of AM-TaaS and Test Scripts written by users. Scripts were written in the Python language and worked together with the *MonkeyRunner* tool. Users can submit one or many test scripts. It could be a single file with "*.py" extension or multiple files that need to be compressed into a single file with the extension "*.zip".

4. EXECUTION EXAMPLE OF AM-TaaS

In this section we describe the application of AM-TaaS in an example. To start the use of the mobile testing service, we need to define the input artefacts. In this case, we used the mobile app "Grades" and for the test script submitted by the user, we have a set of test cases called "fulltestapp2.zip". For this example, the user is already logged in the AM-TaaS framework.

Then, we will describe step by step the use of AM-TaaS, as described in Section 3.2.

Step 1: The user sends the input artefacts (*Grades.apk* and *fulltestapp2.py*) as shown in Figure 9.

Step 2: When the input artefacts are saved, the available emulators/devices to be selected are shown. For this case, we will select the devices "Galaxy Nexus" and "Nexus 6". All tests will be executed on these selected devices. Then, we need to press the button "Select Emulator", as shown in Figure 11.

Step 3: after selecting emulators, the test execution will be started and we need to wait until this finishes. At this moment, AM-TaaS executes all the default test and the user script test over all selected emulators.

¹⁰ <https://developer.android.com/about/dashboards/index.html>

Uploading files

Please upload your Application it need to be a file with
If you have only one file it must have "py" (Python Script)

Session Test Name *

GRADES TEST

Apk File: grades.apk

Script Test File: fulltestapp2.zip

Figure 9. AM-TaaS Second Step: Uploading Files.

Step 4: when execution is finished, the framework presents all results separated by device and by each test script. Among these results, the result of the automated test scripts performed by AM-TaaS are shown and, as in this case the users send their test cases. The results are presented on the Section "User Scripts", as shown in Figure 10 (emulator Galaxy Nexus) and Figure 12 (Nexus 6).


Test Results


Galaxy Nexus (Google API)

OTA INSTALL			
Id	Test	Time	Status
1	Installation	0:00:10.311001	passed
2	No space left on device	0:00:43.737999	passed

OTA UNINSTALL			
Id	Test	Time	Status
1	Uninstall	0:00:15.083999	passed

USER SCRIPTS			
Id	Test	Time	Status
1	Add Class	0:01:22.151995	Passed
2	Add Grades	0:01:09.416973	Failed
3	Edit Grades	0:00:35.920427	Failed
4	Delete Class	0:00:35.973805	Failed

 Device.Log



TEST-LOG

Debug.log

3496 bytes

Figure 10. AM-TaaS Final Step: Results for Galaxy Nexus.

Select emulator devices

Device Name	Brand	Platform	Version	API	Ram	Resolution	CPU	Select
Nexus One (Google API)	Google	Android	4.2.2	17	768	480X800	ARM	<input checked="" type="checkbox"/>
Galaxy Nexus (Google API)	Google	Android	4.2.2	17	1024	768X1280	ARM	<input type="checkbox"/>
Nexus 5 (Google API)	Google	Android	4.3.1	18	1024	1080X1920	ARM	<input type="checkbox"/>
Nexus 6 (Google API)	Motorola	Android	4.4.2	19	1024	1440X2560	ARM	<input checked="" type="checkbox"/>
Nexus 7 (Google API)	Google	Android	4.4.2	19	1024	1200X1920	ARM	<input type="checkbox"/>

Figure 11. AM-TaaS Emulated devices information.

Information displayed on results are the test name, time of execution and the test status (passed / failed) for each test case. For this example, in both devices functional tests were executed (Add and Edit Class, Add and Delete Grade).


Test Results

Nexus 6 (Google API)

OTA INSTALL			
Id	Test	Time	Status
1	Installation	0:00:09.284000	passed
2	No space left on device	0:00:37.917000	passed

OTA UNINSTALL			
Id	Test	Time	Status
1	Uninstall	0:00:14.859000	passed

USER SCRIPTS			
Id	Test	Time	Status
1	Add Class	0:01:15.538411	Passed
2	Add Grades	0:01:36.570550	Passed
3	Edit Grades	0:01:13.347581	Passed
4	Delete Class	0:00:41.526548	Passed

 TEST-LOG
Debug.log
3496 bytes

- Device.Log

Figure 12. AM-TaaS Final Step: Results for Nexus 6.

Moreover, Test results are displayed together with a link to download the log file for each emulator and a link for the test log. These links are shown in Figure 10 and Figure 12 ("Device log"). This file contains the emulator/device's log; this information helps users to take decisions.


```

[Starting Devices]
[Emulator Internal Name] emulator-5554
[Emulator Name] AVD17_NexusOne
160707 16:50:31.938:I [MainThread] [com.android.chimpchat.ChimpManager] Monkey
Getting package and activity name...
Installing com.evernote...
160707 16:50:34.937:I [pool-1-thread-1] [com.android.chimpchat.adb.AdbChimpDev
starting command: monkey --port 12345
160707 16:50:34.937:I [pool-1-thread-1]
...
) Verifying if Apk was installed
*****NEW***** Apk installed successfully
Saving results...
Uninstalling...
com.evernote
com.evernote.ui.HomeActivity
com.evernote/.com.evernote.ui.HomeActivity
Starting App Uninstaller...
160707 16:51:30.545:S [MainThread] [com.android.chimpchat.adb.AdbChimpDevice]
command: pm uninstall com.evernote
Saving results...
Filling memory...
3055 KB/s (2000000 bytes in 0.639s)
2976 KB/s (50000000 bytes in 16.404s)
2807 KB/s (1000000 bytes in 0.347s)
2992 KB/s (10000000 bytes in 3.263s)
Installing Apk without enough memory...
160707 16:52:03.960:S [MainThread] [com.android.chimpchat.adb.AdbChimpDevice]
installing package: INSTALL_FAILED_CONTAINER_ERROR
Verifying if Apk was installed
Saving results...
Freeing up memory space
Installing Apk for User Test...
FINISHED

```

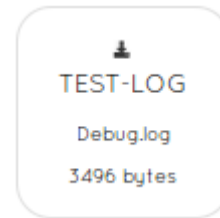


Figure 13. AM-TaaS Final Step: Results (a) Log Test Content (b) Link to download.

The file “TEST-LOG” contains the log of test scripts’ execution for each emulator/device. This information is important because it contains more details about failed tests on each emulator/device. An extraction of this file is shown in Figure 13.

In the next section, we present conclusions and future work to be executed as part of this work.

5. CONCLUSION AND FUTURE WORK

In this paper, we present a summary of commercial mobile cloud testing services and the most used tools to integrate the cloud testing with automated testing. Also, we present the architecture and implementation of AM-TaaS as a way to contribute to mobile app quality using cloud resources, as a testing service.

The AM-TaaS framework is focused on the use of emulated devices due to the high cost to acquire real devices and to perform the test on many devices and automated test scripts as a way to help on the time of execution of each test. Furthermore, all the layers and their components were explained that have been used for implementation and a step by step example of AM-TaaS execution. Using this framework allows users such as testers or developers to apply different types of tests. The current framework allows functional and compatibility testing to be performed over many emulated devices and real devices.

An experimental study is being conducted in order to evaluate the efficiency of AM-TaaS to perform tests on different mobile devices and applications. Also the perceived usefulness and ease of use is being evaluated. In addition, AM-TaaS will be compared with another environment testing for this experimental study.

ACKNOWLEDGMENTS

The authors would like to thank CNPq, CAPES and FAPEAM for their financial support to this research.

REFERENCES

- [1] STATISTA. “Forecast for the number of mobile app downloads”. <http://www.statista.com/statistics/266488/forecast-of-mobile-app-downloads/>. Retrieved on July of 2016.
- [2] STATISTA “Worldwide mobile app revenues in 2015, 2015 and 2020” <http://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast>. Retrieved on July of 2016.
- [3] H. Muccini, A. Di Francesco, and P. Esposito, “Software testing of mobile applications: Challenges and future research directions”, in: *7th International Workshop on Automation of Software Test (AST)*, 2012, pp. 29-35.
- [4] B. Jiang, X. Long, and X. Gao, “MobileTest: A tool supporting automatic black box test for software on smart mobile devices,” in *Proceedings of the AST International Workshop on Automation of Software Test*, 2007, pp. 1–6.
- [5] S. She, S. Sivapalan, and I. Warren, “Hermes: A Tool for Testing Mobile Device Applications”, in *Proceedings of the Australian Software Engineering Conference*, 2009, pp. 121-130.
- [6] B. Kirubakaran and V. Karthikeyani, “Mobile application testing — Challenges and solution approach through automation,” in *Proceedings of the PRIME International Conference on Pattern Recognition Informatics and Mobile Engineering*, 2013, pp. 79–84.
- [7] A. Méndez-Porras, C. Quesada-López, and M. Jenkins, “Automated Testing of Mobile Applications: A Systematic Map and Review”, in *XVIII Ibero-American Conference on Software Engineering*, 2015, pp. 195-208.
- [8] Anthony I. Wasserman. 2010. “Software engineering issues for mobile application development”, in *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10)*, 2010, pp. 397-400.
- [9] A. Domenico, A. Fasolino, P. Tramontana, and B. Robbins, “Testing Android Mobile Applications: Challenges, Strategies, and Approaches”. *Advances in Computers* 89, 2013, pp. 1-52.
- [10] J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, “Mobile Application Testing: A Tutorial,” *Computer* (Long. Beach. Calif.), vol. 47, no. 2, 2014, pp. 46–55.
- [11] C. Tao and J. Gao. “Modeling mobile application test platform and environment: testing criteria and complexity analysis”. In *Proceedings of the 2014 Workshop on Joining Academia and*

Industry Contributions to Test Automation and Model-Based Testing, 2014, pp. 28-33.

- [12] L. Yu, W.-T. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao, "Testing as a Service over Cloud," in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, Jun 2010, pp. 181–188.
- [13] S. Huang, Z. J. Li, Y. Liu, and J. Zhu, "Regression Testing as a Service," in *2011 Annual SRII Global Conference*, 2011, pp. 315-324.
- [14] Y.-H. Tung, C.-C. Lin, and H.-L. Shan, "Test as a Service: A Framework for Web Security TaaS Service in Cloud Environment," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, Apr. 2014, pp. 212–217.
- [15] L. Yu, X. Li, and Z. Li, "Testing tasks management in testing Cloud environment," in *Proceedings - International Computer Software and Applications Conference*, 2011, pp. 76–85.
- [16] M. Yan, H. Sun, X. Wang, and X. Liu, "WS-TaaS: A Testing as a Service Platform for Web Service Load Testing," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, 2012, pp. 456–463.
- [17] J. Gao, W.-T. Tsai, R. Paul, X. Bai, and T. Uehara, "Mobile Testing-as-a-Service (MTaaS) -- Infrastructures, Issues, Solutions and Needs" in *Proceedings of IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, Jan. 2014, pp. 158–167.
- [18] K. Inki, I. Ari, H. Sozer, "A Survey of Software Testing in the Cloud," in *2012 IEEE Sixth International Conference on Software Security and Reliability Companion*, 2012, pp. 18–23.
- [19] G. Candea, S. Bucur, and C. Zamfir, "Automated Software Testing as a Service," in *SoCC '10 Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 155–160.
- [20] X. Bai, M. Li, X. Huang, W.-T. Tsai, and J. Gao, "Vee@Cloud: The virtual test lab on the cloud," in *8th International Workshop on Automation of Software Test (AST)*, 2013, pp. 15–18.
- [21] K. Blokland, J. Mengerink, and M. Pol, *Testing Cloud Services: How to Test SaaS, PaaS, IaaS*. USA: Rocky Nook Inc., 2013.
- [22] P. Mell and T. Grance, "The NIST-National Institute of Standards and Technology- Definition of Cloud Computing," *NIST Special Publication, 800-145*, 2011.
- [23] J. Gao, K. Manjula, P. Roopa, E. Sumalatha, X. Bai, W. T. Tsai, and T. Uehara, "A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, 2012, pp. 464–471.
- [24] D. Amalfitano, A.-R. Fasolino, P. Tramontana, and B. Robbins, "Testing Android Mobile Applications: Challenges, Strategies, and Approaches", in *Advances in Computers* 89, December 2013, pp. 1-52.
- [25] O. Starov and S. Vilkomir, "Integrated TaaS platform for mobile development: Architecture solutions," in *2013 8th International Workshop on Automation of Software Test (AST)*, 2013, pp. 1–7.
- [26] A. S. Al-ahmad and S. A. Aljunid, "Mobile Cloud Computing Testing Review," in *2013 International Conference on Advanced Computer Science Applications and Technologies*, 2013, pp. 176-180.
- [27] L. Murugesan and P. Balasubramanian, "Cloud Based Mobile Application Testing," *ACIS 13th International Conference on Computer and Information Science (ICIS)*, 2014, pp. 4–6.
- [28] Gartner, "Worldwide Public Cloud Services Market to Total \$131 Billion" <http://www.gartner.com/newsroom/id/2352816>. Retrieved in July of 2016.
- [29] Priyanka, I. Chana, and A. Rana, "Empirical Evaluation of Cloud-based Testing Techniques : A Systematic Review," *ACM SIGSOFT Software Engineering Notes archive, Volume 37, Issue 3*, may/2012, pp. 1 - 9.
- [30] R. Mahmood, N. Mirzaei, and S. Malek, "EvoDroid: segmented evolutionary testing of Android apps" in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 2014, pp. 599–609.
- [31] G. S. De Oliveira and A. Duarte, "A Framework for Automated Software Testing on the Cloud," in *2013 International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2013, pp. 344–349.
- [32] A. Malini, N. Venkatesh, K. Sundarakantham, and S. Mercysalinie, "Mobile application testing on smart devices using MTAAS framework in cloud" in *International Conference on Computing and Communication Technologies*, 2014, pp. 1–5.
- [33] O. Starov, S. Vilkomir, and V. Kharchenko, "Cloud Testing for Mobile Software Systems," in *ICSOF 2013 - 8th International Joint Conference on Software Technologies*, 2013. pp. 124–131.
- [34] App Quality Alliance (AQuA) <http://www.appqualityalliance.org> . Retrieved on July, 2016.
- [35] Rhoton, J.; Haukioja, R. Cloud Computing Architected: Solution Design Handbook. Edition 2013. United States.
- [36] "Android Debug Bridge (adb)" <http://developer.android.com/studio/command-line/adb.html> Retrieved on July, 2016.
- [37] "Monkeyrunner" <https://developer.android.com/studio/test/monkeyrunner/index.html> Retrieved on July, 2016.
- [38] "AndroidViewClient" <https://github.com/dtmilano/AndroidViewClient> Retrieved on July, 2016.
- [39] "UIAutomator" <https://developer.android.com/topic/libraries/testing-support-library/index.html>. Retrieved on July, 2016.
- [40] do Nascimento, J., dos Santos, J., & Dias-neto, A. C. (2014). MTControl : Ferramenta de Apoio à Gestão de Testes de Aplicativos Móveis Baseada nas Diretrizes do AQuA. In *X Workshop Anual do MPS (WAMPS 2014)*, pp. 234–241, 2014.
- [41] T. Kim, Y. Choi, S. Han, J. Y. Chung, J. Hyun, J. Li, and J. W.-K. Hong, "Monitoring and detecting abnormal behavior in mobile cloud infrastructure," *2012 IEEE Netw. Oper. Manag. Symp.*, pp. 1303–1310, Apr. 2012.