ABSTRACT

CLOUD PLATFORM FOR RESEARCH CROWDSOURCING IN MOBILE TESTING

by Oleksii Starov

April 2013

Director of Thesis: Sergiy Vilkomir

Department of Computer Science

Mobile application testing and testing over a cloud are two highly topical fields nowadays. Mobile testing presents specific test activities, including verification of an application against a variety of heterogeneous smartphone models and versions of operating systems (OS), build distribution and test team management, monitoring and user experience analytics of an application in production, etc. Cloud benefits are widely used to support all these activities. This study conducts in-depth analyses of existing cloud services for mobile testing and addresses their weaknesses regarding research purposes and testing needs of the critical and business-critical mobile applications.

During this study, a Cloud Testing of Mobile Systems (CTOMS) framework for effective research crowdsourcing in mobile testing was developed. The framework is presented as a lightweight and easily scalable distributed system that provides a cloud service to run tests on a variety of remote mobile devices. CTOMS provides implementation of two novel functionalities that are demanded by advanced investigations in mobile testing. First, it allows full multidirectional testing, which provides the opportunities to test an application on different devices and/or OS versions, and new device models or OS versions for their compatibility with the most popular applications in the market, or just legacy critical apps, etc. Second, CTOMS demonstrates the effective integration of the appropriate testing techniques for mobile development within such a service. In particular, it provides a user with suggestions about coverage of configurations to test on using combinatorial approaches like a base choice, pair-wise, and t-way. The current CTOMS version supports automated functional testing of Android applications and detection of defects in the user interface (UI). This has a great value because requirements for UI and user experience are high for any modern mobile application.

The fundamental analysis of possible test types and techniques using a system like CTOMS was conducted, and ways of possible enhancements and extensions of functionality for possible research are listed. The first case studies prove the work of implemented novel concepts, their usefulness, and their convenience for experiments in mobile testing. The overall work proves that a study of cloud mobile testing is feasible even with small research resources.

CLOUD PLATFORM FOR RESEARCH CROWDSOURCING IN MOBILE TESTING

A THESIS

Presented To
The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Oleksii Starov

April 2013

CLOUD PLATFORM FOR RESEARCH CROWDSOURCING IN MOBILE TESTING

by

Oleksii Starov

APPROVED BY:

DIRECTOR OF DISSERTATION:_______________________________________

Sergiy Vilkomir, PhD

COMMITTEE MEMBER:____

M.H. Nassehzadeh Tabrizi, PhD

COMMITTEE MEMBER:____

Junhua Ding, PhD

COMMITTEE MEMBER:____

Vyacheslav Kharchenko, PhD, DrS

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:

____

Karl Abrahamson, PhD

DEAN OF THE GRADUATE SCHOOL:

____

Paul J. Gemperline, PhD

Dedicated to

My parents and sister,

For all their love and endless support

ACKNOWLEDGEMENTS

First, I am so grateful to the computer science department and my classmates and friends for their help, support, and encouragement.

I owe my deepest gratitude to Professor Vilkomir, who became my mentor, for his invaluable knowledge and the experiences he shared with me. This thesis would not have been possible without his guidance and support.

I am also very thankful to Professor Kharchenko for his never-ending fatherly support and rapid overseas help, brilliant ideas, suggestions, and comments. I offer him special gratitude for promoting and encouraging me to study at ECU.

Many thanks to Professor Tabrizi and Professor Ding for excellent teaching, for serving on my thesis committee, and for always providing great assistance.

Finally, I owe my gratitude to Sealpoint Co. for providing me with an excellent commercial experience in mobile development and for a great and friendly team culture. I am also thankful to the creators of Android OS and smartphones and to all supporters of the modern mobile world and its software quality.

TABLE OF CONTENTS

LIST OF TABLES VIII

LIST OF FIGURES IX

LIST OF ABBREVIATIONS X

CHAPTER 1: INTRODUCTION AND PROBLEM FORMULATION 1

1.1 Introduction to Mobile Testing 1

1.2 The Problem Statement and Objectives 3

1.3 The Novel Multidirectional Testing Concept 4

1.4 The Novel Concept of Integrated Mobile Testing 6

1.5 The Thesis Summary 10

CHAPTER 2: MOBILE TESTING OVER A CLOUD/RELEATED WORK 13

2.1 General Cloud Testing 13

2.2 General Mobile Testing 14

2.3 Analysis of Mobile Testing Services 15

2.3.1 Device Clouds 16

2.3.2 Services to Support ALM 21

2.3.3 Testing Techniques Provided 23

2.4 Standalone Tools 25

2.5 Research Works in Mobile Testing 27

2.6 Combinatorial Testing 30

CHAPTER 3: THE CTOMS FRAMEWORK/REQUIREMENTS AND DESIGN 32

3.1 Analysis of Use Cases and User Roles 32

3.2 Choose of the High-level Cloud Architecture 34

3.3 The CTOMS Node Architecture 38

3.4 The CTOMS Master Architecture 40

3.5 Test Techniques Integration Methodology 42
3.5.1 Scenarios and Formats of Testing 42
3.5.2 Test Automation Environment 45
3.5.3 Types of Testing Applicability 47
3.5.4 Coverage Calculation 48
CHAPTER 4: THE CTOMS FRAMEWORK/IMPLEMENTATION 51
4.1 Implemented Functionality 51
4.2 Notes on the Process of Development 53
4.3 The CTOMS Node Implementation 54
4.3.1 Overall Structure and Deployment 58
4.3.2 Interactions with Android SDK 61
4.4 The CTOMS Master Implementation 62
4.4.1 Google App Engine Deployment 64
4.4.2 Load-balancing for Test Distribution 66
4.5 Master-Node Communication 68
4.6 Notes on Scalability and Universality 72
CHAPTER 5: CASE STUDY 76
5.1 Testing ECU Mobile for Android 76
5.1.1 Main Native App Testing 77
5.1.2 Web Mobile App Testing 79
5.2 Testing LinkedIn for Android 80
5.2.1 Interactions with Photo Camera 80
5.2.2 Multidirectional Testing Demo 82
5.3 Future Research Using CTOMS 83
CHAPTER 6: CONCLUSION 85

REFERENCES 86
APPENDIX A: TEST SCRIPT FOR ECU MOBILE 97
APPENDIX B: SCREENSHOTS OF ECU MOBILE TESTING 99
APPENDIX C: TEST SCRIPT FOR LINKEDIN 104
APPENDIX D: SCREENSHOTS OF LINKEDIN TESTING 107

LIST OF TABLES

1. List of Device Clouds 20

2. Other Research Work in Mobile Testing 28

3. Cloud Architecture Comparison and Selection for CTOMS 36

4. Application of Different Types of Testing 48

LIST OF FIGURES

1. Multidirectional Testing 5
2. Structure of an Integrated TaaS Platform 9
3. Test Stages and Activities with References to Correspondent Cloud Services 16
4. The CTOMS Structure, User Roles and Use Cases 33
5. Architecture of CTOMS Platform as a Cloud System 37
6. A Variant of the CTOMS Architecture 37
7. Architecture of the CTOMS Node 40
8. Architecture of the CTOMS Master 41
9. An Example of the Remote Full Control over a Device Position 46
10. First CTOMS Platform Implementation 52
11. The CTOMS Node Screenshot for General Settings 56
12. The CTOMS Node Screenshot for Configuration of Devices 56
13. The CTOMS Node Class Diagram for Data Level 59
14. The CTOMS Node Class Diagram for Working with Android SDK 59
15. The CTOMS Node Class Diagram for Performing Tests 60
16. The CTOMS Master Screenshot for Device Selection 63
17. The CTOMS Master Screenshot for Testing Tab 63
18. The CTOMS Master Class Diagram for Managers and Test Controls 65
19. The CTOMS Master Class Diagram for Test Techniques Providers 66
20. The Test Distribution Process 67
21. UML Sequence Diagram of Master-Node Interactions 70
22. The Screenshot of T-way Coverage 77
23. The ECU Mobile Revealed Bugs 78

24. The Interaction with Photo Camera during Testing.. 81

25. The Screenshot of a Private (New) Device Testing.. 82

LIST OF ABBREVIATIONS

ADB	Android Debug Bridge	25
ALM	Application Lifecycle Management	16
API	Application Programming Interface	8
CTOMS	Cloud Testing of Mobile Systems	3
GAE	Google App Engine	36
GUI	Graphic User Interface	9
MVC	Model-View-Controller	53
NIST	National Institute of Standards and Technology	30
OS	Operation System	2
PaaS	Platform-as-a-Service	35
REST	Representational State Transfer	44
TaaS	Testing-as-a-Service	2
UI	User Interface	26
UX	User Experience	16

CHAPTER 1: INTRODUCTION AND PROBLEM FORMULATION

This chapter provides an introduction to the main challenges in mobile testing, and then derives the concrete problem statement and objectives of this thesis. Finally, the novelties of the approach are described in corresponding separate subsections.

1.1 Introduction to Mobile Testing

Mobile development is characterized by a variety of applications with different quality requirements. Online application stores, like the Apple App Store [1] and Google Play [2], offer thousands of market-oriented apps—mobile games, utilities, navigators, social networks, and clients for web resources. At the same time, the interest in critical mobile applications is growing. For instance, online banking has evolved into mobile banking [3], mobile social alerts are widely used to report accidents [4] or warn about hurricanes [5], and special apps exist to monitor traffic [6] and help cardiac patients [7]. Augmented reality apps are used for complex navigation and involve a variety of sensors. A new trend is to use smartphones as components for mobile cyber-physical systems because the powerful hardware has a variety of sensors (White et al., 2010). Mobile applications are even being considered to support processes at such critical facilities as nuclear power plants [8].

These trends require high levels of reliability and quality for mobile software systems. They affect testing, in particular, and the whole mobile development process in general. Too often, the mobile development process ends with the submission of a social application to an online store. The aim is to gain a wider audience of users in a shorter time, but this does not guarantee the quality of the product and non-critical bugs are usually accepted. Some surveys have confirmed that mobile developers usually deal with small apps and do not adhere to a formal development process [9]. In contrast, a totally different approach is required for critical or business-critical mobile applications, including mobile clients for trustworthy enterprise systems and solutions; for example, Facebook's iOS app is crucial for maintaining the company's profile and reputation and thus was rebuilt to overcome the poor quality of the first version [10].

To guarantee these mobile applications' reliability and security, sufficient testing is required on a variety of heterogeneous devices as well as on different OS. Android development is the most representative example of how different applications should function amid a plethora of hardware-software combinations [11]. Adequately testing all of these platforms is too expensive—perhaps impossible—especially for small resource-constrained mobile development companies.

Mobile development has a set of distinctive challenges and features. Mobile application testing has some similarities to website testing as both involve validation in many environments (smartphones and browsers, respectively). The general requirements for both types of testing are similar: applications should function correctly, efficiently, and be reliable and secure in all environments. However, mobile testing presents new activities and requires more effort because it includes web applications that work within mobile browsers or hybrid variants wrapped in native code [9]. This testing also involves a large number of possible combinations of mobile devices and OS [11]. Finally, mobile testing involves the use of actual hardware and so testers need additional knowledge and skills such as build installation and crash-log retrieving.

Advanced mobile software processes typically work according to the Agile-based methodology [12, 13] and include usage of build distribution services to assist in testing, analytical services for maintenance during production, and services to obtain a wider range of mobile devices for testing. These services create a large set of testing-as-a-service (TaaS) resources, or supporting web-applications, that use cloud benefits to facilitate the testing of mobile applications and cover a large range of the specific mobile testing needs. These cloud solutions make mobile testers more effective because they provide complex infrastructure and/or services that are not feasible within small developer companies.

The dominant type of such cloud services is a "device cloud," i.e., a service that provides hosting of remote mobile devices and running of tests in the cloud. Existing commercial variants of such platforms became an inspiration for the current study.

1.2 The Problem Statement and Objectives

Mobile testing over a cloud is an extremely important activity that is very hard to research. As was described above, a lot of industrial cloud services exist that fulfill the initial testers' needs, but investigations to increase their effectiveness such as the application of additional testing methods are impossible apart (within separate research labs).

The goal of this thesis is to develop a scalable platform for effective research crowdsourcing in mobile testing. It should serve as a research variant of commercial cloud services that provide access to remote smartphones for testing, and support the participation of geographically separated sites. Crowdsourcing [14] means that several universities or research labs can contribute mobile devices for the shared device cloud and each will have ability to use it for its own studies and investigations.

Additionally, a support of two novel concepts—multidirectional testing and flexible integration of testing techniques—was stated as a main requirement to the platform. These features that can be useful for both industrial and research aims were observed to be absent in existing device clouds. Each concept is discussed in a separate section that follows.

In the end, a Cloud Testing of Mobile Systems (CTOMS) framework was developed to serve as the platform to facilitate the testing mobile applications and mobile testing research. The cloud service provides the ability to run tests on a variety of remote mobile devices (i.e., smartphones). It is based on a heterogeneous networked system that connects operational computers, mobile devices, and databases with applications. This framework is presented as a combination of hardware (smartphones) and software (applications) that allows for different testing directions. For instance, it is possible to test a new smartphone model for its compatibility with mobile applications and to test a new application on different smartphone models.

The CTOMS platform is an integrated Testing-as-a-Service (TaaS) solution with scalable architecture. All core testing functionalities are jointly implemented in one platform over the cloud. This provides the possibility of different use cases and the ease of adding new functionalities, such as non-functional testing or test planning approaches. The implementation of this single system is less complicated than of separate systems for each testing activity. CTOMS can be used as an internal automation solution within development teams.

The current study focuses on Android as a popular mobile platform with distinguished support by many heterogeneous devices. Thus CTOMS currently supports only Android, but has a universal architecture and solutions that can be applied to other mobile platforms.

This thesis serves as the initial research of cloud mobile testing and consequently a lot of additional functionality can be added to CTOMS. Thus, the applications of testing techniques, different testing types, formats, and scenarios are analyzed and possible solutions are described in the form of methodology to enhance the developed framework. All this should help in conducting further research using the CTOMS platform.

1.3 The Novel Multidirectional Testing Concept

This thesis aims to generalize the concept of a cloud service that provides mobile devices for testing. The developed CTOMS framework extends the typical functionality and provides the ability to test OS version updates and new hardware devices against the most popular or/and important applications. It is important to be confident that the legacy mobile applications will still work properly in a new environment.

Figure 1 illustrates the concept of multidirectional testing and shows the three main types of objects in the system: applications (apps), devices (hardware), and versions of OS. CTOMS provides the ability to test each side on/against others; in other words, it provides multidirectional testing from all possible perspectives. All use cases are in high demand. Simple lines show existing services. Dashed lines show partially new use cases (see [15] for an analogue of testing new devices against applications). Bold arrows show totally new functionality.

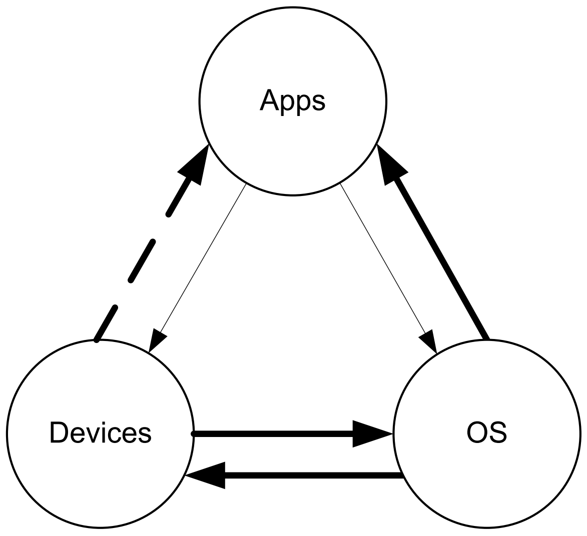


Figure 1: Multidirectional Testing

The current study considers cloud solutions for the following scenarios:

1. Application developers can test a product on different devices and/or OS versions.
2. OS developers can test new versions of an OS on a set of modern devices and the most popular apps to ensure compatibility.
3. Hardware developers can test new device models for their compatibility with the newest OS versions and the most popular applications.

The innovative aspect of this approach is that it provides testing of new OS version against the top popular applications (and test cases used for their development). The relevance of such functionality becomes obvious if we consider how rapidly new versions of iOS or Android systems are released. The same acute situation applies to hardware, thus the device fragmentation testing matrix for Android development can have nearly 4,000 separate Android device models [11].

In general, the need to test OS or hardware against applications is driven by critical systems that contain mobile applications. In such systems, it is very important to guarantee the dependability of crucial applications with a newer version of the OS or a new device. Critical mobile applications must still work properly after OS updates and provide the same reliability, or else the resulting faults can be very expensive. The presence of all testing perspectives in the CTOMS framework provides the ability to comprehensively test mobile systems because the hardware components are also key ones for them.

Android development is the most representative example of the problem of diverse configurations to be tested. The variety of heterogeneous devices, OS versions, screen resolutions, and other parameters is significant. The popularity of the Android OS necessarily makes the question of cloud testing extremely important. As such, the current research focuses on building a version (prototype) of the CTOMS that supports Android testing. At the same time, the offered architectural and implementation solutions are universal in the meaning that it would be easy to tailor the system for other mobile platforms. The proposed CTOMS framework also takes into consideration the possibility of support for different kinds of devices: from smartphones to mobile robots and from microcontroller-based embedded systems to field-programmable hardware [16].

1.4 The Novel Concept of Integrated Mobile Testing

Another new aspect proposed in this thesis is to embed the test model, i.e., appropriate testing techniques for mobile development, within the cloud framework. Specifically, pair-wise testing [17] is considered for this purpose. The general idea to integrate test designs or test generation functions within a cloud service is not something original by itself; this thesis considers the services for mobile testing, and state-of-the-art cloud-based mobile testing indicates the lack of test techniques' integration (see section 2.3.3).

From a user's point of view, the embedding of a testing model operates as suggestions provided by CTOMS, namely, suggestions relating to what hardware-software configurations need to be tested, the testing criterion to choose, the minimal test coverage required, the risk statistics about particular device configurations, etc. As a result, a user can choose an appropriate testing model for a given situation in terms of desired budget, time, requirements, etc.

A testing model must also provide the general organization of testing (i.e., launching tests, storing results, etc.). This provides an opportunity to collect statistics in the system, and for instance, to advise a user that a particular device caused the main part of defects during other similar applications testing.

CTOMS can also be considered for providing reliability, performance, and security testing. In the context of security testing, the following variants of additional services are proposed:

- Implementation of different kinds of static analysis.
- Whitebox approaches based on decompiling Java classes [18].
- Model-driven approaches for security and language-based security analysis.
- Automated stress security testing and fuzz testing.

For performance testing, it is proposed to use frame rate counters similar to Windows phone Emulator [19]. For reliability testing, detailed usage of statistics is proposed in conjunction with the long-term performing of tests.

A detailed design of the CTOMS framework aims to provide the ability to extend functionality with such kinds of testing in a way similar to plug-ins. The global goal is to create a comprehensive testing environment. This environment should present a comprehensive view of the TaaS platform for mobile development. From this point onward, we will consider TaaS for mobile development as a cloud service that provides the following functionalities:

- Testing on heterogeneous software-hardware configurations.
- Embedded application of functional and non-functional testing techniques, including static analysis.
- Usage of statistics as a possible basis for testing techniques and reliability evaluations.
- Test planning and management facilities, including test team operation.
- Multidirectional testing (i.e., testing under different roles: app developer, device, or OS producer).
- Multitenancy and user management.
- Possible inclusion in continuous integration processes or usage through Application Programming Interfaces (APIs).

The last two requirements should ease the use of such systems in more popular ways, especially among mobile developers. Different scenarios may require automated usage, a private or public cloud, or automation with a build process and other requirements.

The consideration of the different TaaS functionalities according to these requirements leads to the idea of a mutually beneficial integrated solution. Figure 2 shows that the proposed core functionality of such a solution will have three main components: a cloud of devices, a static analysis engine, and a statistics sub-system. The desired high-level testing services will use these components separately or jointly when performing their functions. For instance, the dashed line in Figure 2 shows a scenario in which, based on statistics, a particular set of devices is chosen to perform automated testing in the cloud. The cloud of devices can be used for functional or manual testing after the application of some testing techniques to select the proper device coverage or can be used for non-functional testing such as the long-term stress testing of the app on selected devices. Testing techniques can use information provided by the statistics to calculate coverage or the statistics can be used to determine the duration of reliability testing. Static analysis can be applied at any time simultaneously. Finally, a cloud of devices with supporting databases (data storages) can be used to test in different directions using assumed-to-be correct results from previous testing on trustworthy models.

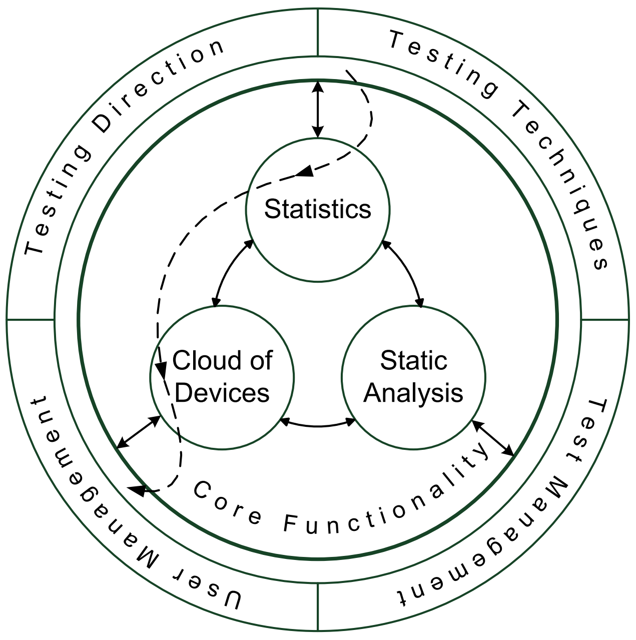


Figure 2: Structure of an Integrated TaaS Platform

Figure 2 shows four groups of high-level testing services built onto a core infrastructure. Thus "testing techniques" means performing functional tests on a set of remote devices such as unit-tests or GUI-based test scripts [20] with or without the application of supporting testing techniques, e.g., combinatorial testing to calculate desired configurations coverage [21]. Non-functional testing is also taken into account and involves the application of performance testing on remote devices using frame rate counters [22], statistical testing, or security testing by conducting static analyses of source codes [18].

"Test management" means the presence of supporting services such as build distribution [23], test plan creation, and testing team management.

"User management" describes the need to separate the users of the system and provide access rules. This includes general user management and global multitenancy requirements.

"Testing direction" means the availability of different perspectives on testing. For instance, a user may utilize a service both as an app developer to test apps on devices and as a hardware or OS developer to test a new device (connected to the cloud on his/her side) or the OS on this device against available binaries and test artifacts.

1.5 The Thesis Summary

This section provides a summary sheet of the current thesis, highlighting the most important points of the study and the work done.

***Problems addressed*:** improvements and extensions to modern cloud mobile testing and organization of the effective research crowdsourcing in mobile testing via a cloud platform.

***The relevance*:** widespread mobile applications, growing popularity of critical apps, weaknesses of typical mobile testing, and ample room for enhancements of supporting cloud services, and the need and complexity of the extensive research in the field.

***Objectives*:** development of the effective research crowdsourcing in mobile testing over a cloud by the creation of the easily scalable lightweight CTOMS framework that supports a convenient way of engaging participants and contributors.

***Novelties*:** (1) implementation in the framework of all possible testing directions (apps against device models or OS versions, devices or OS versions against top apps, OS versions against devices and vice versa) and (2) advanced integration of test techniques with a focus on the coverage calculations.

***Methods*:** (1) leveraging the benefits of cloud computing, meaning the creation of the distributed cloud of real devices (smartphones, tablets, etc.) using the PaaS facilities, (2) GUI-based test automation for functional testing and detection of defects in the user interface, and (3) combinatorial strategies for coverage calculation.

***Technologies used*:**

- Android (Android SDK, including Monkeyrunner tool for test automation)
- Google App Engine (including Blobstore)
- Java, the VAADIN 6.8 framework, RESTful web services
- JPA, the Derby database, embedded Tomcat 7, Hazelcast lock
- The integrated NIST ACTS tool

***Results and Contribution*:** (1) provided in depth analysis of the state-of-the-art cloud-based mobile testing, (2) created a methodology of test techniques application for mobile testing over a cloud, (3) developed the CTOMS framework, and (4) performed two demonstrative case studies.

Tool: CTOMS provides a cloud service to run tests on a variety of remote Android devices using standard Monkeyrunner test scripts that are focused on taken and comparison of checkpoint screenshots. The platform consists of two parts: the master deployed at Google App Engine cloud and a slave node to be deployed at a participant's site (on a server with connected smartphones).

***Publications*:** Results of this thesis were reflected in the two papers accepted at the top international conferences in software engineering. They are the following:

1. Oleksii Starov and Sergiy Vilkomir, "Integrated TaaS Platform for Mobile Development: Architecture Solutions," Proceedings of the Eighth International Workshop on Automation of Software Test (AST 2013), San Francisco, USA,

May 18–19, 2013, in conjunction with the 35th International Conference on Software Engineering (ICSE 2013).

2. Oleksii Starov, Sergiy Vilkomir and Vyacheslav Kharchenko, "Cloud Testing for Mobile Software Systems: Concept and Prototyping," Proceedings of the Eighth International Conference on Software Engineering and Applications (ICSOFT-EA 2013), Reykjavík, Iceland, July 29–31, 2013, as a part of the 8th International Joint Conference on Software Technologies (ICSOFT 2013).

In conclusion to this chapter, we can summarize that the main goal of this thesis is to develop effective crowdsourcing in mobile testing over a cloud. The created CTOMS framework is supposed to be easily scalable and provide multidirectional testing and integration of testing techniques.

The thesis is organized as follows. Chapter 2 provides state-of-the-art mobile testing over a cloud. Fundamental analysis of existing cloud services and a discussion of related work provide justification and refinement for the stated objectives and a background for used solutions. Chapter 3 provides detailed requirements and high-level design of CTOMS. It also describes methodology of test application and possible enhancements of the framework in section 3.5. Chapter 4 deals with implementation of CTOMS, describing non-trivial questions. Then Chapter 5 provides results of the first case studies with CTOMS. Finally, Chapter 6 presents the overall conclusions.

CHAPTER 2: MOBILE TESTING OVER A CLOUD/RELATED WORK

This chapter describes the state-of-the-art mobile testing over a cloud and provides an analysis and review of related works. First, general questions of cloud testing are discussed. Then existing cloud services to facilitate mobile testing are analyzed to justify the desired CTOMS functionality. Finally, research in mobile testing and specifically combinatorial testing techniques that are targeted to be implemented in CTOMS are described.

2.1 General Cloud Testing

Many research papers have stated that testing extensively migrates to the cloud nowadays [24–28]. Reviews and classifications of testing cloud services include solutions for web systems and mobile development [29, 30]. Cloud benefits are used not only to support performance, load, or reliability testing of websites, but also to assist with providing required hardware resources (i.e., remote smartphones) for different needs for mobile testing. Cloud-based mobile testing is a young but very topical issue [31].

The database at the Cyber Security and Information Systems Information Analysis Center provides a large list of cloud testing references [32]. Technical and research issues about testing over the cloud are analyzed in [33] and [34] respectively.

This work uses the term "cloud service" as the most general understanding of cloud computing [35], i.e., cloud service is a software tool or hardware resource that is delivered over the Internet. The definition means that we also take into account such web resources as build distribution solutions and online issue tracking systems. The term "device cloud" (i.e., mobile device cloud or cloud of devices) will also be used, pointing to both the cloud service's nature and the many geographically dispersed devices.

Many specialized studies exist regarding the general architecture and construction of cloud and distributed systems [35, 36], including providing service through application programming interfaces (APIs) [37]. Technical issues for the tests on the cloud are discussed in [34], including Hadoop usage for test distribution. Device clouds (services that provide hosting of smartphones and run tests on multiple remote real devices) require special algorithms for effective test distribution to make overall test execution time as minimal as possible. A comparison of general load balancing algorithms can be found at [38].

2.2 General Mobile Testing

Mobile development has a set of distinctive features and the following specific challenges can be mentioned [9]: support of many hardware and software platforms, correct work with a variety of sensors, interconnections with other applications, high requirements for users' experiences and the quality of the user interface [39, 40], and the existence of web mobile and hybrid applications that incorporate all of these challenges to web development.

Mobile applications are popular among startups and approaches for quick prototyping to evaluate the concept of an app [41] are now in high demand. All of these features contribute to the complexity and specifics of mobile testing [42, 11]. As for mobile testing in this work, I mean comprehensive testing of a mobile system that includes the testing of mobile apps as well as mobile operation systems (OS) and the related hardware. Different investigations have pointed to the required mobility of the apps in terms of their ability to function in different environments and configurations as the root challenge of testing [42].

uTest published The Essential Guide to Mobile App Testing [11], a book that comprehensively and coherently describes challenges and techniques in mobile application testing. A lot of research exists about automation and facilitation of the testing process, including leveraging of cloud abilities [26, 43–48]. Companies that provide cloud services for mobile testing (cloud of devices) usually assist their customers with a set of guides [49, 50].

Examples of testing matrixes to cover all smartphone models or OS versions generate an enormous number of combinations [11]. The issue is significant for the Android platform because of its representatively large number of supported devices with different characteristics (e.g., screen resolution, size of memory, and set of sensors). The problem is compounded by the fact that a smartphone simulator or an emulator cannot fully substitute for the hardware [11]. At the same time, the development for different mobile platforms looks similar. Platforms have similar developer websites with necessary documentation, examples, and suggested patterns [51]. The principles of the application life cycle are similar, for instance, comparing Android to the Windows Phone 7 [52].

Many software development companies are interested in the mobile market and many mobile platforms now exist: Android, iOS, Windows Phone, Symbian, etc. New ones appear regularly like the recent Ubuntu Mobile OS [53]. According to Gartner, Android devices have most of the market [54] and Forbes says that the Android platform aims to meet enterprise requirements in the near future [55]. Previous research on the bug statistics for the Android OS [56] proved that the Android (with Symbian) has effectively organized an open-sourced bug-tracking system that deals with bugs and makes the platform better. The number of applications in Google Play is now more than 600,000 and is increasing steadily [57]. The open source nature of Android makes it popular among the scientific community, and many examples of research studies targeted at the Android system can be found.

2.3 Analysis of Mobile Testing Services

To facilitate mobile testing, various cloud benefits are used and different TaaS, or supporting services, exist. Figure 3 provides references to them, along with mapping to correspondent testing stages. The presented types of testing were partially taken from a diagram on Perfecto Mobile's guide that shows the demanded device allocation during different application lifecycle management (ALM) stages [49]. The diagram was extended by adding conceptualizations as a separate ALM activity, plus concept, security, and user experience (UX) testing, as well as highlighting test activities such as test planning, management, and issue tracking that are all specific to real-life mobile development.

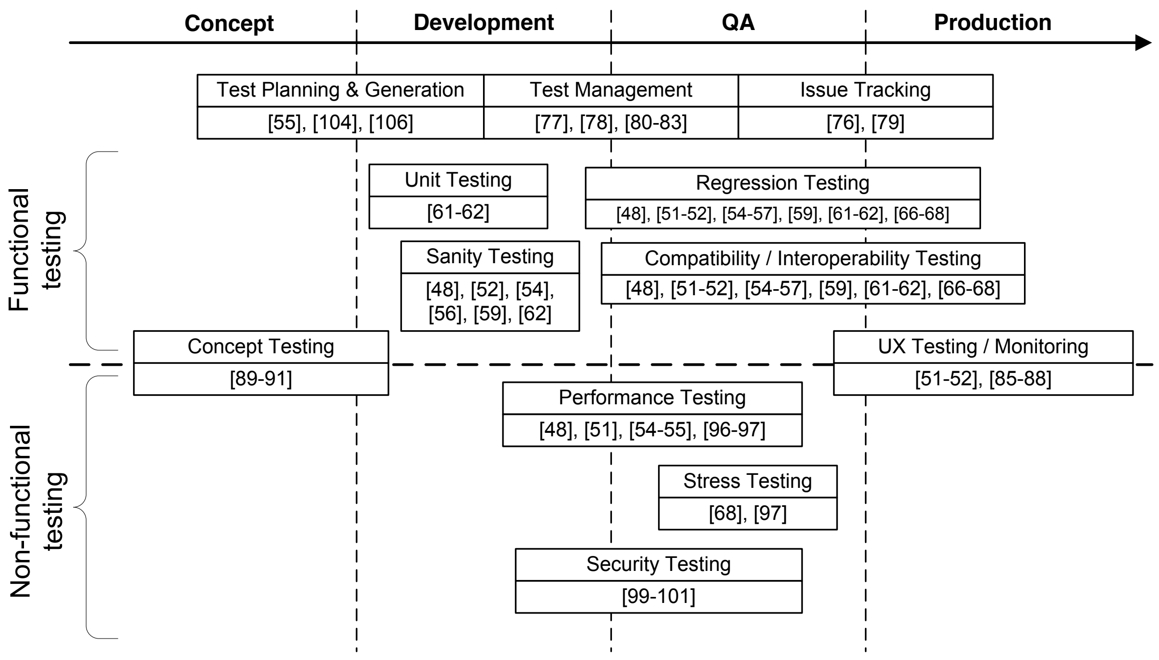


Figure 3: Test Stages and Activities with References to Correspondent Cloud Services

The set of cloud services for mobile testing can be divided into three types: device clouds (mobile cloud platforms), services to support ALM, and tools to provide processing according to some testing techniques. The following sub-sections describe each type separately.

2.3.1 Device Clouds

The majority of cloud services for mobile testing serves as a "cloud of devices" and provides remote access to smartphones in the cloud in order to accomplish testing, in other words, provides device hosting. Such services usually aid mobile developers in using remote smartphones as real devices for manual testing (interactive testing through a web interface), recording of scripts, and automatic running of tests on a range of models. For instance, Perfecto

Mobile service [58] provides all of this functionality representing different modern hardware and software mobile platforms (Android, iOS, Windows Phone, and Symbian) and can be integrated with HP UFT (QTP) [59] or MS Team Foundation Server [60]. Devices available in the system have different parameters, for example, testing different types of Internet connections is possible. The service works with two kinds of test scripts: QTP and the Perfecto Mobile Application. Perfecto Mobile is only a public service, but UFT Mobile [61] can also be deployed as a private cloud. UFT Mobile provides automated functional testing and special solutions for realistic mobile performance testing (e.g., LoadRunner and Performance Center).

Keynote DeviceAnywhere [62] is a similar service that provides online manual and automated testing of a mobile app on a variety of devices. It can be integrated with existing ALM through HP QTP, IBM RQM [63] or special Java APIs.

The SOASTA service [64] provides two advanced solutions: TouchTest test automation for multi-touch, gesture-based applications and CloudTest for scalable mobile application testing (performance or load-testing with millions of geographically distributed emulated users). TouchTest scripts can be recorded and performed against user's own device. Users can control test devices via IP addresses.

The Cigniti device cloud [65] provides remote access to a variety of mobile devices via own proprietary mobile test automation framework, with test accelerators for test automation and performance testing. Cigniti is suitable for network carrier testing.

SeeTest by Experitest [66] provides device cloud that can be deployed as a private platform within an organization. Test automation facilities include test script recording/performing on real devices or emulators and integration with HP UFT (QTP), TestComplete, C#, RFT, Java, Perl, Python. SeeTest also provides manual testing tools.

The CloudMonkey service [67] runs MonkeyTalk scripts across many Android emulators and iOS simulators. Screenshot reports are positioned as the base testing results. CloudMonkey test jobs can be integrated with continuous integration (CI) servers like Jenkins [68].

The Appium on Sauce service [69] covers two functionalities: iOS device hosting and easy CI. The latter means that it can be used as a build server and testers do not need to set up developer environment on local machines. Test automation is implemented with Selenium [70], and interactive testing is only possible for web mobile applications. Appium can be deployed privately.

The TestDroid Cloud [71] is a device cloud service oriented towards Android apps testing that uses the TestDroid AppCrawler engine to verify application devices' compatibility. TestDroid Recorder can be used to generate reusable Android JUnit test cases. Test results consist of screenshots and device logs. A tester can compare screenshots to check for GUI bugs. TestDroid can also be integrated with Jenkins or leveraged through REST APIs.

The Scirocco Cloud [72] has all of the functionality of a device cloud, except of script recording. It supports only the Android platform and provides manual access to remote devices through its HTML5 web interface. Test automation is done by using one of three drivers: AndroidDriver, Monkeyrunner, or NativeDriver. Results are provided as a set of screenshots to compare.

The LessPainfull device cloud [73] is oriented for Android and iOS apps testing. As a test automation engine, it uses Calabash for Cucumber [74] and accepts Cucumber-based test scripts. LessPainfull provides two options: private cloud tailored for single customer and shared cloud with devices common for several customers.

TestQuest [75] is a distributed framework for deployment within an organization. It is oriented towards Android application testing and can be integrated with MS Visual Studio.

The ZPX service provides device hosting and mobile test automation in the cloud [76] and is compatible with HP ALM products.

Jamo [77] provides a set of tools to perform remote and scheduled testing on a device. For instance, Wanconnector in combination with Remote Device Screen provides access to a device within different geographical locations. The M-eux Test tool supports web application testing.

Apkudo's device analytics [78] provide some elements of multidirectional testing by testing devices (e.g., new smartphone models) against the top 200 apps from the market [15]. Similar services are available for smartphone hardware testing, but these have no relation to mobile apps like Datum [79] that provides verification of calls, data quality, and video quality. Apkudo also offers free public and fully automated stress testing of the Android applications on the big range of models using the Monkey tool [80].

Table 1 summarizes the device clouds mentioned above and a comparison based on supported mobile platforms, types of testing, and delivery type of cloud solution. Manual testing means the remote operation of a device via a web interface, and automated testing incorporates functional and regression testing and different kinds of automation. All device clouds provide compatibility testing as intended. Public cloud means service with shared devices, while a private cloud means an infrastructure allocated to a single user or a system to be deployed on a user-developer's site.

Two known research attempts within universities to create and investigate test-bed cloud solutions for mobile development are SmartLab [81] and the Android Tactical Application Assessment and Knowledge (ATAACK) Cloud [82]. Both are distributed systems that connect a set of mobile devices under the Android OS for application investigation, development, and testing.

Table 1: List of Device Clouds

Cloud Service	Supported Platforms			Types of Testing	Delivery Type	
	Android	iOS	Other		Public	Private
Apkudo [78]	+			Stress (automated), New device approval	+	
Appium on Sauce [68]		+		Manual for web applications, Automated	+	+
Cigniti [65]	+	+	+	Automated, Interoperability, Performance, Network	+	
CloudMonkey [67]	+	+		Automated, UI-oriented	+	+
DeviceAnywhere [62]	+	+	+	Manual, Automated, Monitoring, Coverage	+	+
Jamo [77]	+	+	+	Automated		+
Perfecto Mobile [58]	+	+	+	Manual, Automated, Performance, Monitoring	+	
Scirocco Cloud [72]	+			Manual, Automated	+	
SeeTest [66]	+	+	+	Manual, Automated, On a new devices		+
SOASTA [64]	+	+	+	Manual, Automated, Load, Performance, Gesture-based	+	+
TestDroid Cloud [71]	+			Automated, UI-oriented, On a new devices	+	+
UFT Mobile [59]	+	+	+	Automated, Load, Performance, Monitoring		+
Zap-Fix [76]	+	+	+	Automated		+

The SmartLab is an experimental test-bed being developed at the University of Cyprus. It provides more than 40 connected Android smartphones plus emulated devices, but not many details are described or known.

The ATAACK Cloud is new joint project for Virginia Tech, the University of Maryland, and Vanderbilt University, with the support and funding by Air Force Research Laboratories. Its goal is large-scale mobile application testing and investigations.

These research studies consider device clouds with several smartphones connected to one computer (vertical) and several computers with connected smartphones (horizontal) scaling of devices, i.e., fully distributed systems, and how to provide access and testing.

Many studies regarding less-scaled test frameworks for distributed mobile testing [83] that are not cloud services and many tools for vertical-scaled test automation only [84] exist, but their reviews are beyond the scope of this chapter.

All services mentioned in this section appear in Figure 3 with the following logistics: services that support the running of unit tests listed under “unit testing,” services that support online manual testing listed under “sanity testing,” references to script automation techniques of these services listed under “regression testing,” all cloud devices listed under “interoperability/compatibility testing,” and references to special integrated non-functional test approaches of these services as listed under correspondent types of testing (see section 2.3.3 for examples).

2.3.2 Services to Support ALM

The application lifecycle management of mobile applications has own specifications and many cloud services exist that support test-related activities within ALM. Several examples of these cloud services are listed below.

1. Mobile developers, like all software developers, use issue tracking systems, e.g., with Agile-oriented plugins [85], more complex solutions like IBM Rational Quality Manager [86], or test management systems like TestRails [87]. Some of these are integrated with software configuration management and facilitate code reviews or code style checks [88]. A review of similar tools and solutions is not the goal of this work, so Figure 3 shows only several base examples.

2. Mobile testing involves the use of actual hardware and so testers need additional knowledge and skills, such as build installation or crash-log retrieving. To facilitate beta build distribution activities, many cloud services exist [89–92]. Some of them provide functions for test team management [89] or build provisioning and deployment to the store (AirOnApp for iOS

[90]). TestFlight service [89] helps to deal with the iOS build management and distributes them via email between separated testers. It provides an easy application installation on a real device, i.e., by a tap on the link in an email opened on a smartphone. A similar service for Android is Launchpad [91]. The HokeyApp [92] build distribution provides extended functionality to collect live crash reports, feedback from users, and analysis of resulting test coverage. Usage of these services for build distribution can be integrated along with the continuous integration process of the company [93] (e.g., via job scripts for the Jenkins build server [70]).

3. User experience testing and monitoring of an app in production are required activities within mobile testing. Several analytics services gather usage statistics like [94] and these can be incorporated in a mobile app. Perfecto Mobile service also provides some solutions for monitoring performance [95].

The following two services incorporate user experience testing in the build distribution facilities. The UserTesting service [96] provides many real users who will examine an app and provide feedback about their experience with the app and thoughts about it. The Amazon A/B testing for Android [97] provides a service that distributes two builds that differ in some features between two unique groups of users. Then it provides measurements and results about which feature is more successful.

4. Mobile development is very popular among startups and usually requires rapid prototyping for concept feasibility evaluation. Thus such services exist like [98] to easily create interactive prototypes, or [99] to share an app demo, or [100] to create realistic mockups. All of these are needed to test the concept and idea of the app (i.e., if it can hit the market) at a minimal expense.

2.3.3 Testing Techniques Provided

This section discusses testing techniques on existing cloud services for mobile development. Device clouds described in section 2.1 provide different techniques for test automation (recording, distribution, and execution). This includes unit tests and GUI-based testing. Examples of approaches are standard Android SDK tools Monkeyrunner [101] and Monkey [80], special solutions like SOASTA TouchTest, and solutions based on object recognition (e.g., Eggplant automation based on VNC technology [102]).

Test automation has its own weak sides, and according to experts in the field, cannot serve as a total substitution for manual testing [103]. The issue that was noticed during the analysis of cloud test automation was the delivery of the test input data to mobile sensors (GPS, accelerometer, camera, etc.). While solutions to send dummy GPS coordinates exist, situation with a photo camera is more complicated because it requires the simultaneous changing of a picture (preferable physically in front of a camera) while performing a script. A variety of mobile apps use a camera as a part of their key functionality (e.g., shopping apps and QR code readers [104]), and proper testing requires test cases with snapshots from different distances, angles, lights, etc. Other problematic aspects of automation are the sophisticated (approximate) screenshots comparisons, executions of direct device-to-device communication during the test, and others.

Device clouds provide compatibility, interoperability, and regression testing. Many services provide embedded tools to support performance monitoring and load testing [58, 59, 64, 65] or even automated stress testing on a variety of devices [78].

There are special cloud services that aid with mobile performance and load testing. For instance, SandStrom [105] can be used for load testing of web mobile applications and NeoLoad [106] focuses on load testing of back-end servers by emulating typical mobile devices working in parallel and sending appropriate content to the server. There are also standalone solutions for test techniques applications like performance frame counters on Windows Phone Emulator [22] that theoretically can be leveraged in a cloud.

Security testing is mainly presented by static check techniques. Checkmarks [107] provides scanning of source code and supports Android and iOS applications. Mobile App Security and Privacy Analysis by Veracode [108] scans and evaluates binary files for vulnerabilities and can be leveraged through APIs. Another type of services exists based on experts. For instance, uTest experts will assist with mobile security testing by manual penetration and using internal static and dynamic security testing solutions [109]. Other solutions to guarantee mobile security focus on proper development processes according to secured methodologies and approaches [110]. At the same time, research papers about novelty mobile security testing approaches exist (that potentially can be leveraged by some cloud services) [18], but these are not described in the present review.

Concept testing, UX testing, and monitoring techniques were comprehensively described in section 2.3.2 as parts of services that support ALM.

Mobile testing services should incorporate test planning and test generation techniques. This review indicates the lack of such functionality. Only Keynote DeviceAnywhere Test Planner [111] provides a coverage calculation for smartphone models to test that can be considered as application of combinatorial testing techniques, but it can be extended by using pairwise [21], t-way [112], or other approaches. HokeyApp only provides test coverage monitoring and analytics, i.e., the matrix of the devices and languages that were tested. Cigniti Test Advisory Services and TestRails provide more high-level test planning and control facilities.

The situation with cloud services for mobile testing is changing extremely rapidly: new ones appear and old ones get new functionalities. Thus, it is hard to guarantee that the provided list of tools and services is exhaustive, but it can serve as a useful baseline.

2.4 Standalone Tools

Any mobile platform has a correspondent software development kit (SDK) for app developers. Usually the producers of mobile platforms provide developers with a debugger, emulator or simulator, plugin for popular IDE, etc. The toolsets for Android, iOS, or Windows Phone development are very similar. Each platform also provides similar development support. For instance, web developer portals provide similar guidelines on how to use the available tools. In this section, we describe the most important standard tools (i.e., available from SDK) for Android app testing and several third-party extensions or analogues.

The basic tool for working with Android devices is *Android Debug Bridge* (ADB) [113], which is a command-line utility to control Android devices. Device detection, debugging, execution of shell commands, and access to a device's file system is possible by using ADB. A high-level development environment like Eclipse (with the Android Development Tools plugin installed) implicitly uses ADB to install and debug builds within a connected device.

Android SDK provides two special tools for the GUI-based automated testing of applications. The first is *UI/Application Exerciser Monkey* [80] for GUI stress testing, which generates a set of pseudo-random user events and sends them to an Android device. Previously, the Apkudo service [78] was mentioned to provide a cloud of devices for long-term stress testing of an app using Monkey. It shows the statuses of the application being tested on each device, i.e., it either crashed after a sequence of random events or it is still running. Crash logs and other supporting information are provided.

A more advanced tool for automated testing provided by SDK is *Monkeyrunner* [101], which runs on test scripts written in Python with several special classes available to provide support of touch, press, type, drag events, shell commands, intent invocations, app installations, and removal. Functionality is sufficient for basic GUI-based automation. So the following two strategies of interaction with interface components can be used: (1) dynamic coordinates calculation (screen sizes can be dynamically retrieved) and (2) components enumeration through focus change. At the same time a tester who writes test scripts should remember to put in appropriate delays (or special workarounds) between long-term events or actions and the results check. Monkeyrunner is suitable for screenshot analysis, as it provides methods to take screenshots during test script checkpoints and compare them. Thread-safeness is not guaranteed, but test scripts can include efficient simultaneous launches on several connected devices (and thus screenshots can be taken from several smartphones at the same time). This feature is important for CTOMS implementation and will be referenced in the following chapters.

An *AndroidViewClient* extension [114] exists for Monkeyrunner that enables more high-level test scripts, particularly to address UI components in a test script by name or text. But this library only supports "rooted" devices with ViewServer installed [115] or newer devices with UIAutomator [116] (Android API 16 and greater). *UIAutomator* is part of the Android SDK revision 21 and up and comes with the UIAutomatorViewer tool that lists all the UI objects.

The current version of the CTOMS framework uses the Monkeyrunner tool for test automation. Related implementation questions will be discussed later. This choice was made because the tool is provided by SDK producers and is simple enough for the first CTOMS implementation. This tool gives us the ability to focus more on novel cloud architecture and the services of the framework.

Robotium [117] is another popular engine for the automated testing of Android applications. It is an extension of the Android test framework (JUnit tests for Androip applications) used to write easy and powerful automatic black-box tests. Similarly, the Robolectric is based on JUnit 4 and runs Android tests directly on the JVM. Both of these tools point to another direction, i.e., the application of unit tests for mobile testing and even GUI-testing.

Other test automation solutions exist. Previously, several cloud services that provide a run of tests on multiple real devices were mentioned as having their own solutions for test automation. For instance, LessPainful [73] accepts test scripts written in Cucumber using Calabash-Android [74].

All of the aforementioned test automation drivers can be used in CTOMS. One of considered enhancements is to provide users with a choice of test scripts to use. The principles of usage are similar to Monkeyrunner, so it does not require a lot of work to integrate another driver like Robotium.

2.5 Research Works in Mobile Testing

Previous sections provided a review of existing services and tools. To complete the state-of-the-art mobile testing, related research works should be analyzed. Table 2 summarizes the descriptions of related research papers and studies. Each of them concerns a testing aspect that can be used in the cloud. For instance, many of the research studies deal with test automation, and theoretically, any service like device clouds can use described approaches as the test automation driver. In the same way, such extensions like test generation or static analysis can serve as an additional functionality integrated within any cloud service to facilitate mobile testing.

Table 2 shows research areas and contributions for papers and highlights the year of release and the targeted mobile platforms. We can conclude that the popularity of mobile testing continues to grow and touches all possible aspects from effective test generation and design to execution and monitoring. At the same time, Android became the most popular platform under study. An open-source nature, prevalence in the market, support of an enormous number of devices, and ease of development (no provisions or jailbreaks are needed as in the case of iOS)—all make it the choice of researchers.

These listed studies are potential directions for implementation in the integrated cloud service like CTOMS. They do not discuss cloud solutions for mobile testing, but instead present actual issues and techniques and describe possible supporting functionality.

Table 2: Other Research Work in Mobile Testing

Year	Ref.	Mobile Platform	Research Area	Contribution
2012	[118]	Multi (shown for J2ME)	Automation of mobile app testing	Framework that does not require a device under testing to be connected to a computer
	[18]	Android	Whitebox automated security testing of mobile apps	Fuzz test generation approach/testbed for emulation in the cloud
	[119]	Android	Automatic categorization of mobile apps	New method for categorizing Android applications through machine-learning techniques (while accepting malicious apps into the market)
	[120]	Android	GUI-based unit testing of mobile apps	Framework to test applications from GUI
	[121]	Android	Testing mobile apps through symbolic execution	Application of symbolic execution to generate test cases for mobile apps
	[122]	Android	Verification of touch screen devices	Test environment and supporting Android app to test touch screens
	[123]	Android	Automated mobile app testing through GUI-ripping	Technique and real-life case study of bug detection
2011	[124]	Android	GUI crawling-based testing of	Technique for rapid crash testing and

			mobile apps	regression testing
	[125]	Multi	Model-driven approach for automating mobile app testing	Tool suite to apply Domain-Specific Modeling Language
	[126]	Android	Automation of mobile app testing	Review of the Android Instrumentation and the Positron frameworks
	[20]	Android	Automation of mobile app testing	Approach to use the Monkey tool in conjunction with JUnit
	[127]	Android (Dalvik)	Automated privacy testing of mobile apps	Automated privacy validation system to analyze apps (while they are accepted into the market)
	[128]	Multi (shown for Android)	Automation of service-oriented mobile app testing	Approach for decentralized testing automation and test distribution
	[129]	Android	Model-based GUI testing of mobile apps	Extensive case study
	[130]	Multi	Automated test case design strategies for mobile apps	Comprehensive review of challenges and correspondent techniques
	[131]	Android	Static analysis of mobile apps	Extensions to Julia system to provide formally correct analysis of mobile apps
	[132]	Android	GUI unit-testing of mobile apps	Techniques to assess the validity of the GUI code
2010	[133]	Multi (shown for Android)	Adaptive random testing of mobile apps	Test case generation technique
2009	[134]	Windows Mobile	Automated GUI stress testing of mobile apps	Review/automated GUI stress testing tool
	[135]	J2ME	Automation of mobile app testing	Tool for testing mobile device applications
	[136]	Multi	Automation of mobile app testing	SOA based framework for mobile app testing
2007	[137]	Symbian, Windows Mobile	Automation of mobile app testing (black-box)	Event-driven tool for test automation
	[138]	Windows CE	Remote evaluation of mobile apps	Tool for remote usability evaluation
2004	[139]	Multi	Simulation and execution of distributed mobile apps	Proposed special uniform workbench

2.6 Combinatorial Testing

Application of combinatorial approaches to mobile testing can aid in dealing with large amounts of different combinations of hardware and software parameters that should be covered by the tests. Coverage calculation is a crucial activity within mobile testing. According to [51], there are nine families of Android OS presented in the market (not counting lower sub-versions and correspondent builds without Google APIs), four types of screen resolutions (small, normal, large, and extra), and four levels of screen density. Other parameters like type of Internet connection (WiFi, 3G, or 4G), size of RAM, vendor, and a processor's characteristics should also be taken into account to provide adequate coverage during testing.

Many combinatorial testing materials can be found on the corresponding webpage of the National Institute of Standards and Technology (NIST) [140]. One of the simplest and easiest ways to implement combinatorial approaches is the Base Choice [141]. The idea is to create a base test case that represents the most important (common or popular) value for each parameter, and then create others by varying the value of only one parameter at a time. The base test case can be created using statistics, especially in case of mobile testing (i.e., what screen resolution is the most spread or what vendor shares the best part of the market).

Pair-wise [142, 143] and t-wise (t-way) [144] testing are the most common and powerful combinatorial testing approaches [141]. According to the t-wise testing approach, for each subset of t input parameters of a system, every combination of valid values of these parameters should be covered by at least one test case. In pair-wise testing, which is a case of t-wise testing, t equals 2. The idea behind the t-wise approach is that the faults in the software are more likely triggered by a small number of input parameters, with the benefits being that t-wise testing providing reasonable coverage of software input space while using a small number of test cases. For example, if there are 15 Boolean input variables, the total number of various input combinations is 215 or 32,768. However, it takes only 10 input combinations (as pair-wise test cases) to cover all of the different values for each pair of input variables.

Some examples of combinatorial tests based on different configurations of Android application can be found in [21]. Other similar techniques, including t-wise testing [145], MC/DC [146], and RC/DC [147] testing criteria are also considered for integration with CTOMS.

The ACTS tool created by the NIST [148] provides an engine to calculate different combinatorial strategies. The developed CTOMS framework leverages this tool in a similar way to the Hexawise web resource [112], but provides a web interface tailored for mobile testing purposes. At the same time, other combinatorial testing tools exist [149] and can be leveraged in CTOMS, but ACTS was proved to be usually faster and produce less number of test cases [148].

More details about the application of combinatorial strategies for mobile testing and their implementation in CTOMS will be discussed in Sections 3.5 and 4.1, respectively.

In conclusion, we can summarize the two weak sides of existing cloud services for mobile testing. First, the lack of integrated testing techniques like combinatorial testing for coverage calculation, test generation services, and services for automate dynamic security testing were noted. Second, technical challenges in test automation within device clouds were mentioned, highlighting the issue of mobile sensors' control during test execution.

The current thesis is aimed at dealing with both of these issues by providing correspondent analysis and methodology of test techniques application and by development of the CTOMS framework that can be enhanced to support all desired types, formats, and techniques of testing.

CHAPTER 3: THE CTOMS FRAMEWORK/REQUIREMENTS AND DESIGN

This chapter aims to provide detailed requirements for the CTOMS framework by focusing on desired use cases and possible user roles (from app developers to smartphone producers and vendors). All of these factors affect the high-level design and architecture of the platform.

The architecture of a networked system such as CTOMS can vary significantly in levels of complexity. For example, the size of the desired distributed solution is a dependent factor. CTOMS can be architected as a single PC computer with connected smartphones or as a comprehensive cloud solution that operates hundreds of PC nodes. To achieve all of its goals, CTOMS should be implemented from a large-scale perspective and must correspond to all cloud computing features, such as service delivery, scaling, virtualization, elasticity, multi-tenancy, load balancing, universal access, etc. The cloud type can also differ from a public SaaS solution to a private Infrastructure as a Service (IaaS), with the ability to provide a low-level configuration.

Selected architecture solutions are described in the current chapter after use cases and user roles are defined. Finally, an analysis of the test application is provided to evaluate the flexibility of the chosen architecture and present a methodology for future enhancements.

3.1 Analysis of Use Cases and User Roles

The conceptual structure of the CTOMS framework contains several layers of mechanisms and functionalities. Each should be analyzed, architected, and implemented in a final comprehensive system. Figure 4 illustrates these layers and their connections with possible use cases:

1. The contributor of the device only invests in hardware by connecting it to the cloud system.

2. The application developer uploads the application under test (source codes or binary), specifies test cases (automated scripts or unit tests), gets results as pass/fail statistics or screenshots for checkpoints, manually accesses selected devices for debugging, etc.
3. The device producers test new devices against the base of apps and OS software in CTOMS. Test scripts available in the system for chosen apps can be used. Producers can also specify their own test scenarios.
4. The OS producer connects the devices with new OS versions to the cloud or uploads update packages to the database. Then the new OS versions are tested with apps/scripts/ devices in the system. OS producers can also specify their own test scenarios.

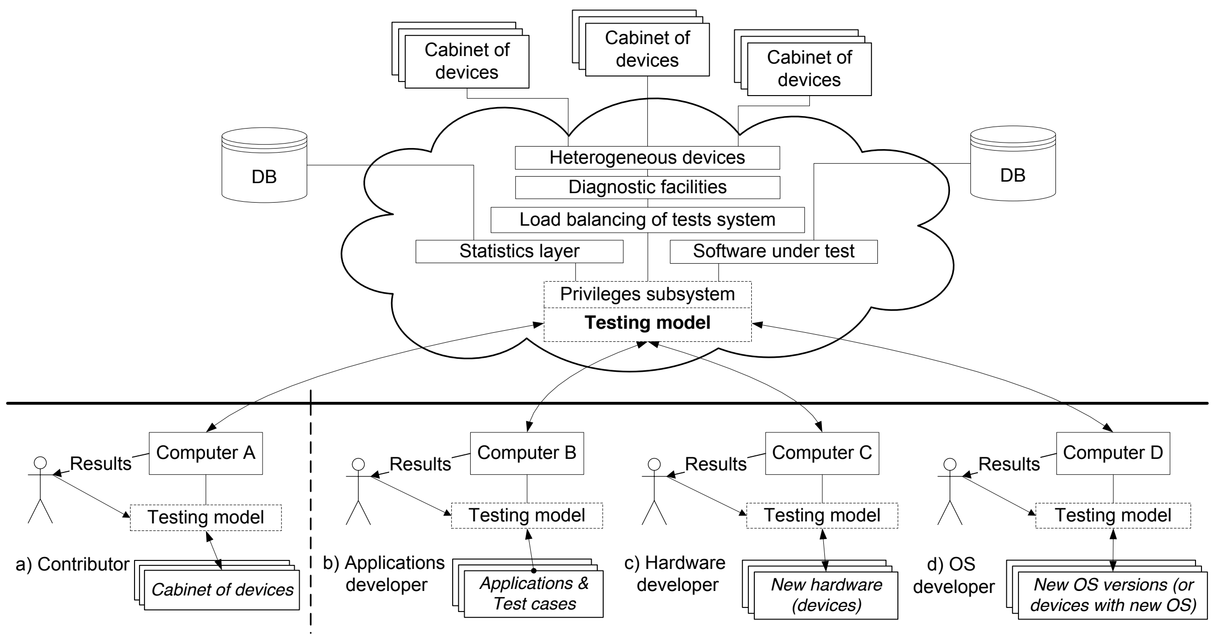


Figure 4: The CTOMS Structure, User Roles and Use Cases

Figure 4 shows that each user can provide a "testing model" while using the framework. For application developers, this means specifying a test strategy, namely, what tests need to be performed on what devices and how. For example, they can specify their own test scripts; select devices, OS versions, coverage, methods to check if a test is passed, etc. For other users, this means not only specification of test strategy or rules of testing (in case of contributor), but also how to perform the testing on connected devices. For example, the settings can be made for which or how many applications to test. The technical interface of the system for contributors, as well as OS and hardware producers, is almost the same.

The innovative feature of CTOMS is its testing model that serves as an internal mechanism and additional service for users (layer inside cloud on the figure 4). This aspect and general testing techniques application are described in more details in the section 3.5.

The databases in CTOMS store the software (applications and OS versions), the testing results, the statistical information about testing, and user information for granting privileges based on the billing and for providing multi-tenancy, etc.

This work with devices requires the development of three additional layers: load balancing of tests execution, diagnostic facilities, and heterogeneous device connections. All of these layers (inside the cloud in figure 4) should be investigated in accordance with earlier specified new use cases, along with the creation of corresponding methodologies. Load balancing of tests execution system means algorithms to distribute test cases between connected smartphones in optimal ways with respect to time of operation and wait time of other users. Simultaneously, general scalability should also be taken into account. All these questions are discussed from the implementation point of view in the chapter 4.

3.2 Choose of the High-level Cloud Architecture

The high-level architecture of the current CTOMS implementation satisfies the requirements of desired integrated TaaS solution. From an architectural point of view, CTOMS is a distributed system that can be implemented in different ways. Architecture depends mainly on the organization of a dominant “cloud of devices” core feature. Other core sub-systems (e.g., static analysis and statistics) are simpler web applications that do not require special load balancing or synchronization of data but, of course, can be deployed to some cloud infrastructures. The following list presents the variants of CTOMS's possible implementation from a cloud-less to a cloud-full solution:

1. Single server. This system consists of the one web application that operates connected devices and/or emulators, manages a local database, and provides extended web interface to end-users. Devices can be connected by TCP (e.g., Wi-Fi) or through USB. Of course, such a solution is not scalable and has crucial limitations on the number of operated devices.
2. Several servers. This system represents master-slave architecture, i.e., one server serves as the master that provides a web interface to end-users. Others serve as slaves or leaf nodes that operate devices (henceforth, terminology "node" will be used). Node computers perform a series of tests, gather statistics, and provide the results to the master. Such a solution does not use the power of cloud computing to increase efficiency of testing. For instance, it will be slow just in case many requests are made at the same time.
3. Master application in the cloud. This solution is similar to a previous one, except the master application is deployed to some cloud (e.g. Platform-as-a-Service, PaaS). In this case, the system leverages the power of auto-balancing for master application and auto-replication for its data storage. Usage of adequate algorithms for test distribution should guarantee a satisfactory speed for the service.
4. Cloud infrastructure. This distributed system is built from the infrastructure level with the aim of providing desired functionality. In other words, algorithms to distribute tests (i.e., load balancing of the test execution layer) are integrated into

the infrastructure. This is the highest level of cloud nature that should guarantee the highest efficiency of work.

This thesis advocates the third variant because it has cloud nature with perceived adequate efficiency and is feasible for implementation by a small research team. The table 3 presents detailed comparison and justification of the selection. Four parameters were considered for the four variants of architecture listed above: ease of scaling, cloud efficiency, complexity of implementation and economical issue, i.e. price of solution. Variants that satisfy certain feature within the current study have 1 in the correspondent column, otherwise 0 (weights of each parameter are approximately equivalent within the scope of the current study). In the result, the chosen variant 3 has the greatest overall value, 4.

Table 3: Cloud Architecture Comparison and Selection for CTOMS

Architecture	Scalability	Cloud efficiency	Complexity	Budget	Overall
Variant # 1	0	0	1	1	2
Variant # 2	1	0	1	1	3
Variant # 3	1	1	1	1	4
Variant # 4	1	1	0	0	2

Figure 5 illustrates the architecture of CTOMS according to the third level of distributed system implementation. This approach separates the presentation and the platform and the information logical layers of the cloud solution, which is similar to the notation used in [35].

The platform layer is represented by the master application, e.g., on Google App Engine (GAE) cloud [150], and optional Hadoop instances layer that leverages the MapReduce algorithm to distribute a test between nodes and the gather the results. Hadoop instances can be organized on other clouds (e.g., Amazon EC2). The master application provides the presentation to the end-user and to slave node computers.

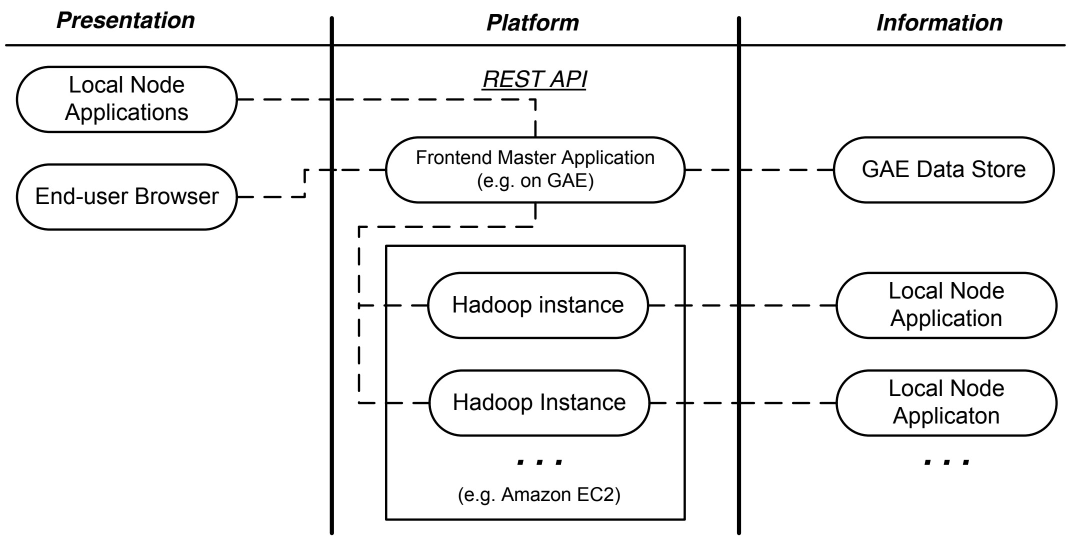


Figure 5: Architecture of CTOMS Platform as a Cloud System

The information layer is represented by the cloud data storage. We use notion data storage instead of a database to highlight that a system also needs to efficiently keep files (binaries, source codes, test scripts, screenshots, etc.), so some cloud storage can be used.

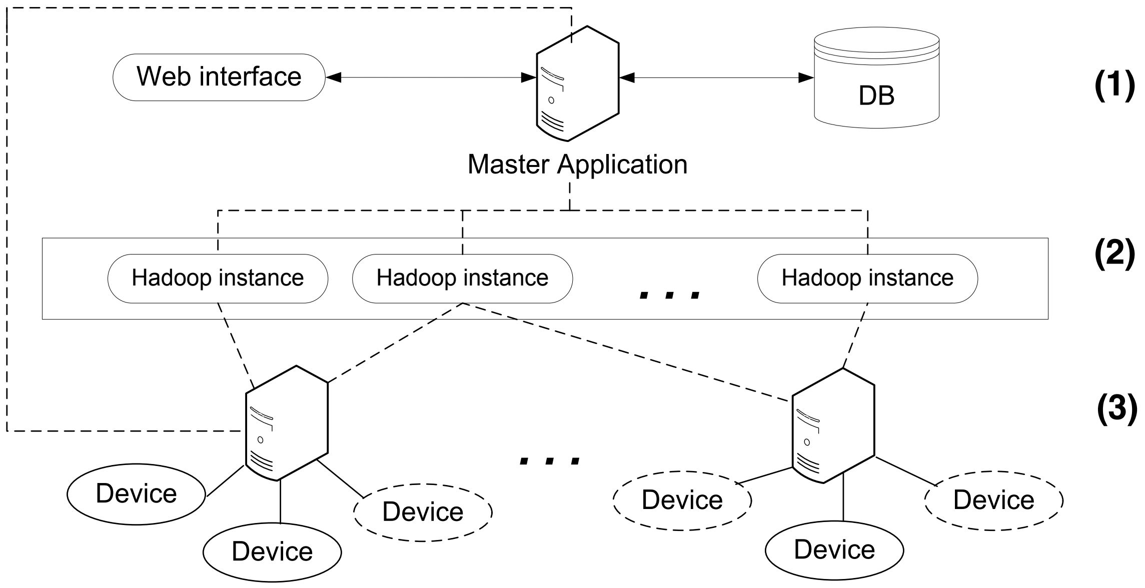


Figure 6: A Variant of the CTOMS Architecture

Figure 6 illustrates the same three-level architecture highlighting the separate components. Communication between subsystems through the Internet (web services) provides a convenient scaling of layer 3. Slave node (leaf) computers with smartphones can be easily connected to a master application in the cloud. They could also operate emulators (virtual Android devices marked with a dashed line). The interface for working with emulators and devices is the same.

The following two sections describe design of the CTOMS application that works on the node computers and the CTOMS master application that works in the cloud respectively.

3.3 The CTOMS Node Architecture

Node application serves as a slave server application. It is supposed to be installed on the contributor's site to operate connected devices. The main responsibilities of a node are managing devices, performing tests, and general settings of interconnections with the master.

Figure 7 illustrates the current architecture of the CTOMS node application. The figure shows that the node provides two interfaces: a web interface to the local user and a web services tier to communicate with the master application. The first interface is used for settings, device information adjustments, and local application testing to check the system's serviceability. Corresponding sub-systems are discussed below.

The device manager uses SDK through the device driver to retrieve information about connected devices. The present CTOMS version is targeted at Android testing only, so Android SDK [113] is used, but the architecture is universal and specific components can be substituted (e.g., to update the current structure for the Windows Phone platform, another SDK can be used during system implementation). The user (administrator of particular node computer) adjusts the device information (i.e., vendor, size of memory, type of Internet connection, etc.) to send to the master application. At the same time, the user can choose active devices for local test sessions. The device manager keeps all information in the internal (embedded) database.

The testing manager uses the testing driver to operate the testing tools, e.g., tools for GUI test automation using scripts [101]). The testing manager consists of two supporting components: test preparation utility and test queue. The first component is used to adjust the test scripts or test commands, (e.g., to parallelize them to run simultaneously on several connected devices; such an approach is implemented in the current CTOMS version for Monkeyrunner scripts [101]). The second component is used to establish the order of performed test sessions. The testing process uses a local file system to prepare test scripts and save temporary screenshots, etc.

The settings manager operates with general settings: the path to SDK, the URL address of the master application, etc.

Figure 7 shows that the architecture of a CTOMS node has a cross structure based on total reuse of the internal components. This means that the same functions can be performed during local testing by the user-administrator or by a request from the master application.

Additional attention must be given to the web services tier. Two variants of the organization of the interconnections with the master application are possible:

- Web services are exposed through a public IP address. This means that the node must be accessible through the Internet.
- The logic of universal periodical calls from node application to master must be determined.

Each of the variants has pros and cons. While the first one is usually complicated to establish, the second one increases Internet traffic and the load on the master application and decreases the service speed. The current CTOMS solution uses the first variant because of economic reasons. The GAE offers a free service within limited usage of the cloud that includes quotes on the number of http requests. Thus the second solution could be expensive in the case of multi-nodes and long-term usage. Supporting approaches such as localtunnel [151] and forwardhq [152] are widely used during development to expose the local web server to the Internet

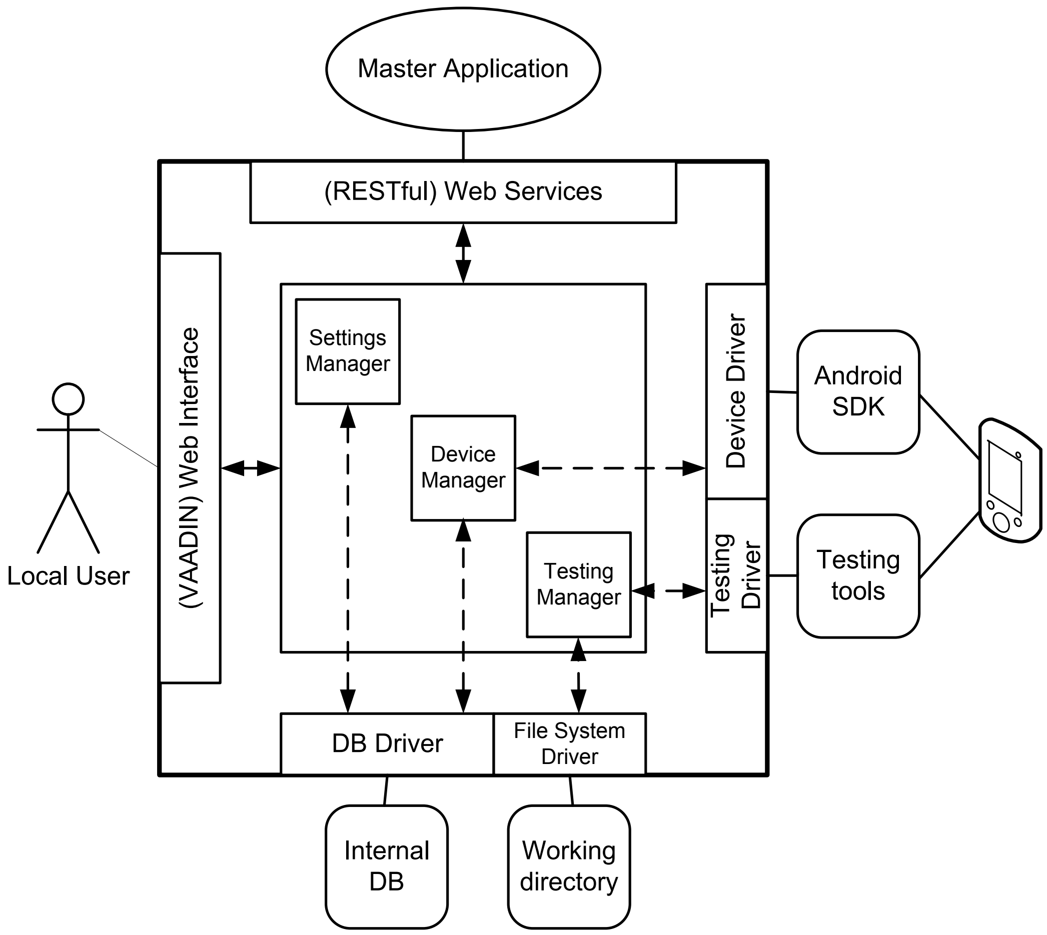


Figure 7: Architecture of the CTOMS Node

3.4 The CTOMS Master Architecture

The master application of the CTOMS system aims to organize the core functionality of the integrated TaaS solution. It collects information about the whole system, organizes the work of nodes, and gathers testing results from them. These functionalities represent the “cloud of devices” core sub-system. In addition, the master application keeps statistics and artifacts in cloud data storage and can contain static analysis facilities. The architecture discussed in this section does not reflect static analysis because it requires another fundamental study, but instead shows the integration of functional testing techniques (i.e., one of the higher-level layers). Figure

8 shows the architecture of the current CTOMS master application. The architecture like the CTOMS node contains two interfaces (presentation layers): web interface to end-user and web services to communicate with slave nodes. The proposed minimal list of web services should contain the following groups:

- Nodes info. Service that node application invokes to send information about available devices and general settings such as password for node.
- Load check. Service that master application invokes from node to get information about test queue (i.e., load on the slave server).
- Testing process. Set of services to send testing task (binary, test script, list of devices, etc.), to get test results and upload/download artifacts.
- External APIs. These serve as interfaces to external tools (services), which provide calculations for some testing techniques. For instance, the current CTOMS implementation uses a NIST ACTS tool wrapped into APIs (but as an embedded library) to build the coverage of configurations.

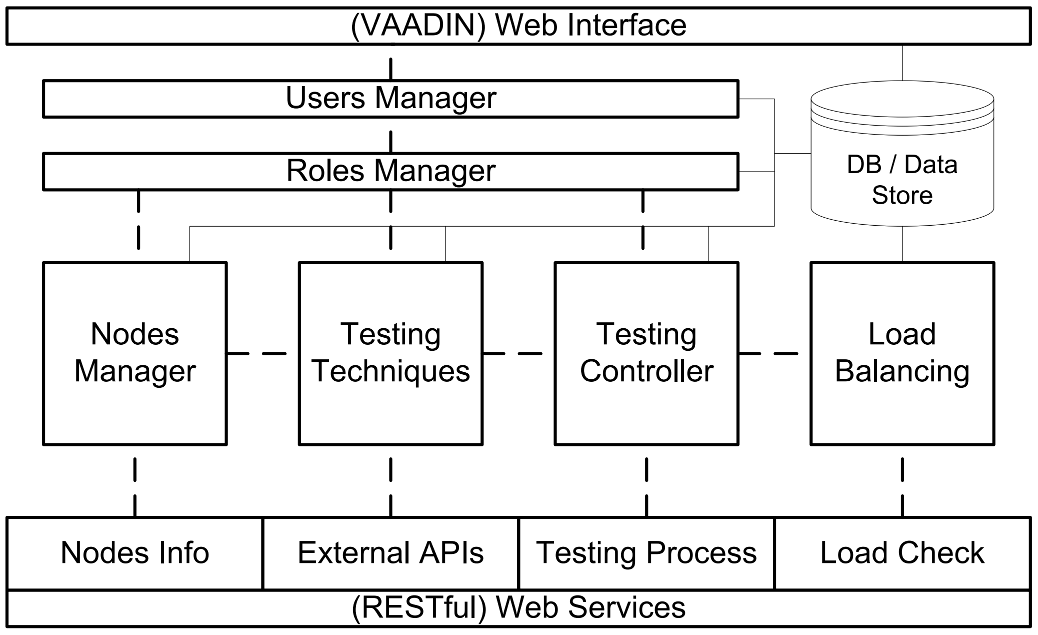


Figure 8: Architecture of the CTOMS master

As for the number of web services groups, four main components exist in the system: nodes manager, testing techniques provider, testing controller, and load balancing controller. Each of these uses cloud data storage (database), but mostly the testing controller because it manages many test artifacts. Users manager and roles manager layers are above the working components and provide service settings depending on the current user and his/her chosen role (perspective).

The next issue is load balancing that is described in details along with implemented solution in the next chapter.

3.5 Test Techniques Integration Methodology

The CTOMS framework represents a core platform that can add many layers of additional functionality, i.e., with different ways and approaches of testing. For instance, different test automation drivers can be used, static analysis of applications can be integrated, or manual access to smartphones through a web interface can be provided, etc. The idea is that the CTOMS framework will be enhanced by crowdsourcing. Different researchers will add new functionalities, if required by the scope of their studies.

This section provides an analysis of possible enhancements and a methodology of test applications using CTOMS. It attempts to cover all possible directions, including possible formats and scenarios of testing, requirements for the test environment, and types of testing. All of these topics are analyzed, given the needs of modern mobile testing, in the following sections.

3.5.1 Scenarios and Formats of Testing

Support for different scenarios of testing is covered by the implemented multidirectional testing concept. General mobile testing includes verification of each of its components: mobile applications, mobile OS, and mobile devices. The following list details the possible use cases:

- An application can be tested against a device or an OS version by using special test scripts provided by testers (or somehow generated).
- A device model can be tested against an app or an OS version running test scripts that were used to test this app or an OS version on an approved model and then comparing the results.
- An OS version can be tested against an app the same way as a device model, and it can be tested against a device, e.g., running scripts without a binary apk-file.

Aforementioned use cases are implemented in the current CTOMS version (see section 4.1 for details). For instance, a user can test his private device connected to the system against some public test scripts and available trustworthy correct results. This can be a new smartphone model or an older approved device with a new version of Android installed. At the same time, user can test just a functionality of OS uploading no Android app during test session.

Automate test generation was not considered as a part of CTOMS, but some services to facilitate a tester's work can be integrated. For now on, additional work was done to accept regular test scripts. The benefit is that users can upload to CTOMS test scripts that they use inside their companies (for local testing or within continuous integration process). Only small requirements to the structure of a test script remain, but the script parallelization (adjustment to run it simultaneously on specific devices in the system) is automatically done by CTOMS (see section 4.3.2 for details).

The following formats of testing are summarized for the current study as follows:

- Automated testing (with test scripts)
- Manual testing (full access to a device)
- Back-end server testing

Automated testing means usage of test scripts that will be run on manually selected smartphones or those selected according to some criteria. Different test scripts are possible, including GUI-based or unit-testing, key-driven or data-driven scripts, etc. Different frameworks can be leveraged like Monkeyrunner (current solution), Robotium, Calabash-Android, etc. At the same time, the service can be extended with script recording, assistance of script generation, or other supporting functions. Results can be delivered as screenshots, video recordings, crash-log reports, console dumps, pass/fail sheet (especially in case of unit tests), etc.

Manual testing means accessing a smartphone or a tablet through a web interface (HTML5) with full screen broadcasting. Users can simulate touches and other interactions with a device using a local PC mouse and a keyboard. Several commercial services exist with similar functionality [58], but for the current research this complex implementation functionality was not interesting.

Back-end server testing means focusing on testing interactions of client mobile apps with their servers. The best part of mobile applications in the market serves as a client and is powered with cloud servers and databases. Their communication (typically through web services like REST APIs or Protocol Buffers) should be thoroughly tested for reliability, performance of servers, correctness of data, security transactions, etc. In this case, full emulation, or usage of a real device, is not required, but emulation of real-life smartphone app behavior is demanded (plus possibly some static analysis or even model-checking of interactions).

Implementation of these scenarios and formats has specific requirements within the test environment. This aspect is analyzed in the next subsection.

3.5.2 Test Automation Environment

This subsection attempts to describe the requirements for devices in the cloud like CTOMS and specifically the environment in which they are placed. By environment, I mean all facilities for performing tests whether automated execution or manual access.

General (implied) requirements are powerful node-servers, an available Internet connection (WiFi or mobile Internet on some devices), devices connected through a USB (bud possible variant of access through TCP), and set to debug mode (including disabled screen auto lock), etc.

Given all available functionalities of modern mobile applications (devices and OS), we present the following list of main features: leveraging of sensors, multimedia, interactions with other applications, and peer-to-peer device communication. Testing needs of each such aspect are analyzed separately.

Leveraging of sensors like an accelerometer, gyroscope, GPS-location, and photo-camera are the most complex parts. Remote testing of such hardware and mobile applications that use this hardware is a big challenge that requires a cloud platform to provide all possible manipulations for a device. Solutions to send mock GPS-coordinates exist [113], but simultaneous changes of accelerometer parameters or pictures in front of the camera while processing a test script require a lot of additional functionality from the test environment. Here, we consider the following possible general directions towards a solution:

- Location (GPS)—Mock locations can be enabled on an Android device (and easily on an emulator [113]), but should be synchronized with the processing of a test script.
- Accelerometer/gyroscope—Mock source code classes require access to code or special hardware solutions should be provided.
- Complex gestures and multitouch—Advanced test script recording is required.

- Camera—A special default camera app is required that shows specified images (not a full solution), streaming of user's web camera (seems infeasible for now), and hardware solutions.

The current CTOMS version implements a variant to work with a camera like a demonstration of the concept. It provides the ability to specify in a test script the URLs of images that will be accessible from the default camera app in Android system (see section 4.1 for details). It enables testing the mobile applications like QR-code readers or social networks, along with the functionality to post images or upload profile photos, but only when using the default camera app.

At the same time, this described approach that worked with a camera during testing does not take into account the real physical parameters of specific cameras. For applications with elements of object-recognition [104] or augmented reality, the whole process of taking a picture from a camera should be tested (different angle, distance, and lights). All of these factors, in combination with the requirements to fully control the device and its position and all six degrees of freedom and environment (especially panorama), leads to the idea of full 3D broadcasting of a device to a tester via a web interface (see Figure 9). Obviously, advanced hardware solutions are required, e.g., an electronic hand to physically rotate a smartphone.

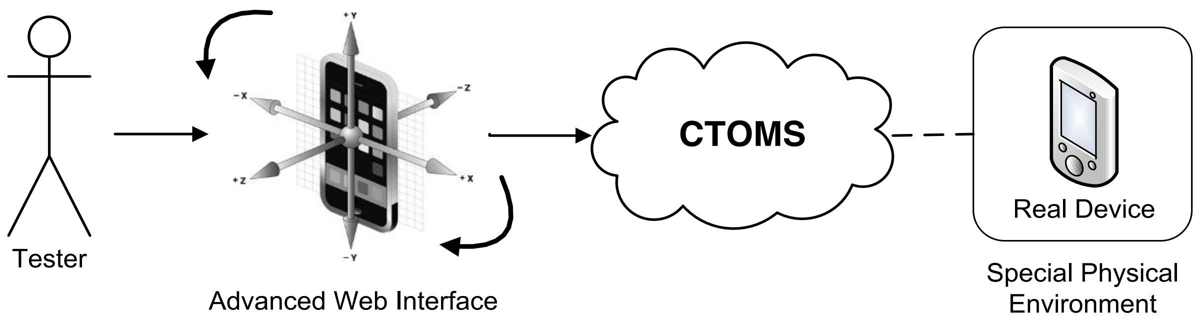


Figure 9: Example of the Remote Full Control over a Device Position

Multimedia features should be tested with full recordings of video, music, and animations. This is a crucial part for games and mobile applications with advanced graphics. An exhaustive test report will provide the ability to detect all possible glitches, even during small effects, transitions, and actions.

Interactions with other applications. Many Android apps have some dependencies on other systems, default applications, or components. Interactions with different versions and sets of system apps must be thoroughly tested. The following list summarizes possible enhancements to the functionality of CTOMS:

- Requirements for a test device to contain preinstalled applications (like system Google Maps, Facebook or Twitter)
- Ability to specify dependencies on preinstalled applications in a test script
- Ability to specify several applications that should be tested at once

The last aspect is that there are mobile applications that rely on peer-to-peer device communication, e.g., games through Bluetooth. Testing these interactions require that the test environment provides the ability of simultaneous tests of an app on several devices. Thus, test scripts should be more powerful in supporting the descriptions of such distributed tests.

3.5.3 Types of Testing Applicability

This subsection deals with the question of the coverage of possible demanded types of testing. Obviously, many different testing needs can be requested by a specific situation. Table 4 summarizes possible solutions for the main kinds of testing that can be implemented in a cloud platform like CTOMS.

Thus, the current CTOMS version implements functional and UI testing through GUI-based automation and provides compatibility, interoperability, and regression testing. More details will be described in Chapter 4.

Table 4: Application of Different Types of Testing

Type of Testing	How can this be represented in CTOMS?
Functional & UI	Any GUI-based automation, manual access, gathering, and analysis of UI metrics and UX statistics
Performance, load, stress	Performance monitoring on both mobile client (e.g., frame counters) and back-end server (e.g., response time)
Security	Static analysis and dynamic fuzz testing (for both mobile client and server back-end)
Compatibility	Means testing on a variety of devices
Interoperability	Means usage of other applications and software components
Regression	Means ability to rerun tests and record test scripts

3.5.4 Coverage Calculation

As was stated in Section 2.6 mobile testing requires a sophisticated coverage calculation. The whole set supported by mobile application device models with heterogeneous technical characteristics and OS versions with different sets of features cannot be exhaustively tested. Instead, a systematic criterion is needed. The idea of this thesis is to integrate testing coverage techniques within a device cloud like CTOMS. The focus was made using combinatorial approaches.

First, the *base choice* approach was implemented. The following three variants of base case formation were considered:

- Settings provided by the administrator manually and taken from statistics of popularity.
- Automate requests to public APIs with statistics about mobile smartphones and Android like AppBrain [57].

- Base case derived from smartphones presented in the system.

In the current CTOMS version, the first variant is activated. The administrator provides which value of each parameter is the most dominant, i.e., the most popular in the market. At the same time, other variants can be easily enabled.

CTOMS currently considers the following device parameters: vendor (producer) of the hardware, OS version (Android API level), type of Internet connection, screen resolution, size of RAM, and processor. The smartphone model, as a parameter, is not used because it represents a one-to-one relation, i.e., a device identifier. CTOMS can have several similar models connected, but embedded algorithms will use the only one for testing that is less loaded at the moment (see Section 4.4.2). Obviously, more characteristics can be found, but these are admittedly the main ones that require verification. It was decided to omit localization (and just specify it for manual selection of the device to test on) because too many values are possible (EN, FR, RU, etc.) making it hard to manage with any combinatorial technique. Typically, localizations that must be supported by a mobile application can be tested locally and do not require a device cloud.

The CTOMS platform was set up with the list of the most popular mobile device producers [49], all Android OS families [153], groups of popular screen resolutions [154], and examples of processors present in the market. The list of Internet connection types contains WiFi, 3G, 4G, and none. The size of the RAM is an arbitrary parameter that should be specified by a node administrator.

Second, the *T-way* testing was integrated by embedding the NIST ACTS tool into CTOMS [148]. In this case, another strategy to provide users with the ability to select exact lists of parameters and correspondent values was chosen. Thus, users can select all parameters they are interested in covering (from a possible six), refine lists of correspondent values, and select the value of t for calculation. More details and demonstrative screenshots will be provided in Chapters 4 and 5.

In conclusion to this chapter, we can state that a number of particular testing needs can be added to the CTOMS platform, and with time, it can evolve into really large enterprise system with many blocks of functionality. This is an additional aim of crowdsourcing using CTOMS—not just to use the cloud for separate investigations, but also to enhance the framework.

The analysis provided above and the listings of possible solutions, directions, and workarounds can be used as a small methodology on how to develop CTOMS for deeper mobile testing research studies. The following chapters will show the current implementation solutions and the first case studies using CTOMS.

CHAPTER 4: THE CTOMS FRAMEWORK/IMPLEMENTATION

This chapter describes the concrete implementation solutions of the current CTOMS version. Starting with implemented functionality, it provides details of how key design and architectural questions were solved. Possible enhancements for next versions and further improvement directions that require separate investigations are also discussed. Finally, conclusions provide a summary of the empirical evaluation of the created cloud system.

4.1 Implemented Functionality

This section describes the functionality of the actual CTOMS implementation more comprehensively. The available version of CTOMS was developed as a feasibility study with the following goals:

- To meet technical risks given the resource-constrained research team
- To create a demonstrative version of the main concepts
- To obtain a platform that suffices for the first case studies with the flexibility to be extended and/or modernized

Figure 10 illustrates the created system and points out that the end-user can operate both the node application on the local computer and the master application deployed in the cloud. The node application provides use cases more typical to the contributors of the devices (i.e., the participants in the crowdsourcing). The master application provides the final desired functionality of the system for mobile app developers. The possible role of the device/OS developer is more complex and requires the user to connect the target device to a node and to organize its testing through the master.

The following key variants of usage are implemented. Mobile apps automate testing using Monkeyrunner test scripts. Monkeyrunner is the standard tool of Android SDK that provides automated execution of GUI-based tests written in Python [101]. A user in the role of a mobile developer uploads binary and test files, which results in a set of screenshots taken from different devices for comparison. Functionality is available on both master and node applications. Such an approach appears easy to implement and mainly targets functional and user interface bugs. Additionally, a default device camera app can be used during tests, instead of a real camera, viewing of a screen with a text field to enter the URL of the target image and a button to snap it. Such functionality demonstrates the leveraging of mobile sensors during testing (see Section 3.5.2 for details).

This CTOMS version implements the application of combinatorial testing techniques. The master application provides functionality to test mobile applications according to selected coverage of configurations criterion: base choice and t-way testing. Under these "configurations," the system understands the possible combinations of parameters, such as device producer, version of Android OS, screen resolution, type of Internet connection, and memory size, etc.

As for multidirectional testing, in the master application, the user can specify him/herself as a device or OS developer and check one (or several) of privately connected devices (or the OS installed on this device) against chosen applications. In the results, the user receives screenshots for comparison with information collected during previous testing of these applications (i.e., with Oracle screenshots that are marked as "correct" in the system). At the same time the user is not required to use an app for testing, only test scrip is required that can interacts with OS and test it.

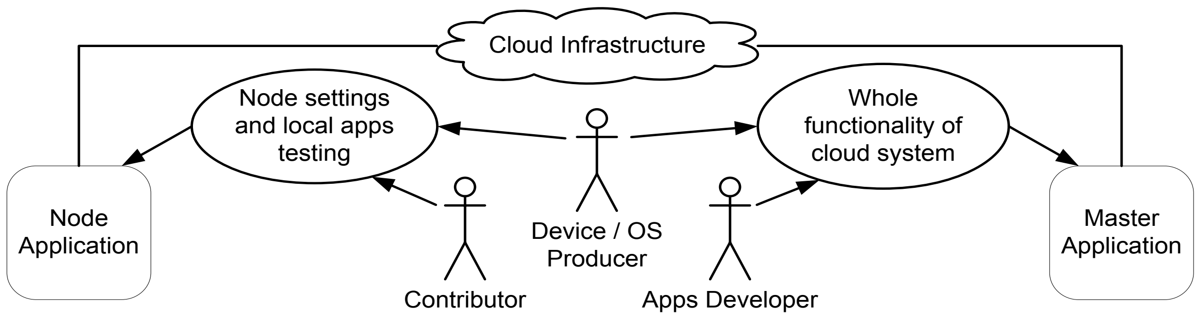


Figure 10: First CTOMS Platform Implementation

The following sections describe the CTOMS Node and CTOMS Master, respectively. The details are provided from UI to source code solutions.

4.2 Notes on the Process of Development

The CTOMS was developed according to the iterative process with SCRUM-like meetings with the thesis director in the role of a targeted user. The following main iterations were performed:

1. A feasibility study was conducted on the utilities to work with Android SDK (ADB and Monkeyrunner) were developed. Other alternatives were analyzed.
2. The CTOMS Node implementation includes the simplified interface of local testing, i.e., finished functionality.
3. The CTOMS Master implementation involves mocked nodes and dummy data.
4. Implementation of the Master-Node communication covers the whole web services tier.
5. Testing, bug fixing, and demo case studies processing were performed.

The following sections describe CTOMS deliverables according to the logic of the iterations listed above. We start from the CTOMS Node that provides the core functionality of test automation and interaction with devices. The CTOMS Master transforms the platform to the real distributed system by providing high-level cloud interface, scenarios, and algorithms. It leverages nodes with a special protocol and agreement of communication.

To finish the brief description of the process used, it is worth mentioning the software engineering methods that were used:

- Separation of UI, business logic, and work with data; elements of Model-View-Controller (MVC) pattern

- Several-steps code reviews and refactoring, application of the trustful Java patterns, and self-documented code
- Verification on Mac OS and Windows platforms
- Implementation of an extended logging within the Master application in the cloud
- Adhering to all guidelines for GAE and VAADIN projects [155]
- Usage of a simple Dropbox service [156] to keep project versions (full software configuration management and version control not required because of a single developer)
- Final black-box testing covering all use-cases and different flows

Usage of the more systematic approaches like unit testing (JUnit), usage of Git, and some issue-tracking system, JMetter and other testing tools will be required with the crowdsourcing evolution of the platform.

To summarize, this subsection provides some brief metrics to describe the size of the CTOMS project and the efforts put into it. Four months were spent to finish all iterations by working periodically in the evenings. The overall estimation is considered to be one month of full-time labor (to create CTOMS with sufficient level of quality).

In the results, the created CTOMS version has the following sizes:

- CTOMS Node has more than 4,500 lines of code that cover more than 50 classes.
- CTOMS Master has more than 9,500 lines of code and more than 100 classes.

4.3 The CTOMS Node Implementation

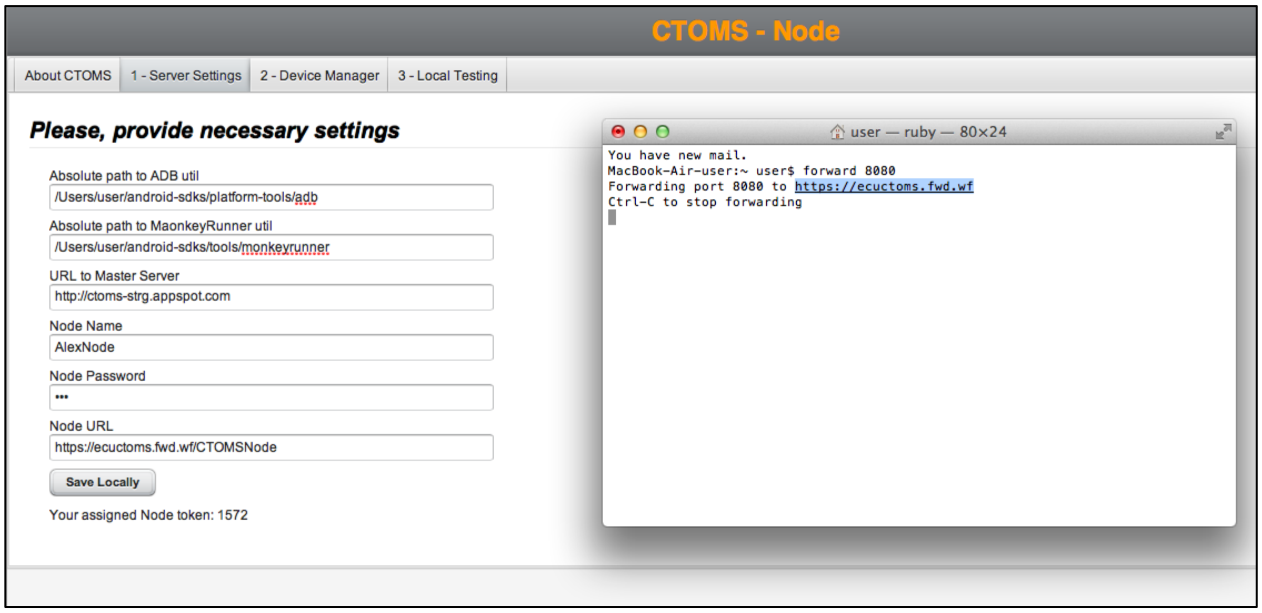
This subsection lists the technical notes on the actual CTOMS node implementation. The process of the CTOMS Node configuration is explained. Figure 11 shows the general settings that an administrator must provide for correct work:

1. Absolute paths to ADB and Monkeyrunner tools from Android SDK if they are not added to system PATH
2. URL to chosen master server (e.g., the current http://ctoms-strg.appspot.com)
3. General passphrase to use CTOMS system (you need to get one from support)
4. Chosen node name and passphrase that will be used to restrict access for your private devices
5. Public URL of your node (Figure 11 shows usage of such solutions like Forwardhq [152])

The next step of a node set-up is to configure a device and register the node in the system. Figure 12 shows a list of automatically detected devices connected to the node and the entered values. The CTOMS Node also provides a unique device token and guesses the model. The following information should be provided for each device:

1. General flags if the device is active, shared, used for local testing, etc.
2. Model parameters include the vendor (producer), Android API level, screen resolution, size of memory, and other including specifying localization (see Section 3.5.4).
3. Flag if advanced AndroidViewClient tests are supported for a device; this is an extension for Monkeyrunner. Requirements can be found here [157], including the presence of AndroidViewClient library on a node.
4. Flag if CTOMSFakeCam is installed as a default app on a device and can be used in test scripts.

For the second item, it is suggested to use "Get possible values" button to retrieve official values (approved by CTOMS Master) for device descriptions and to select from them. Custom ones also can be provided, and the administrator of the CTOMS Master will have the opportunity to approve them as new ones. This approach is necessary to mitigate the problem of different spellings for the same parameters.



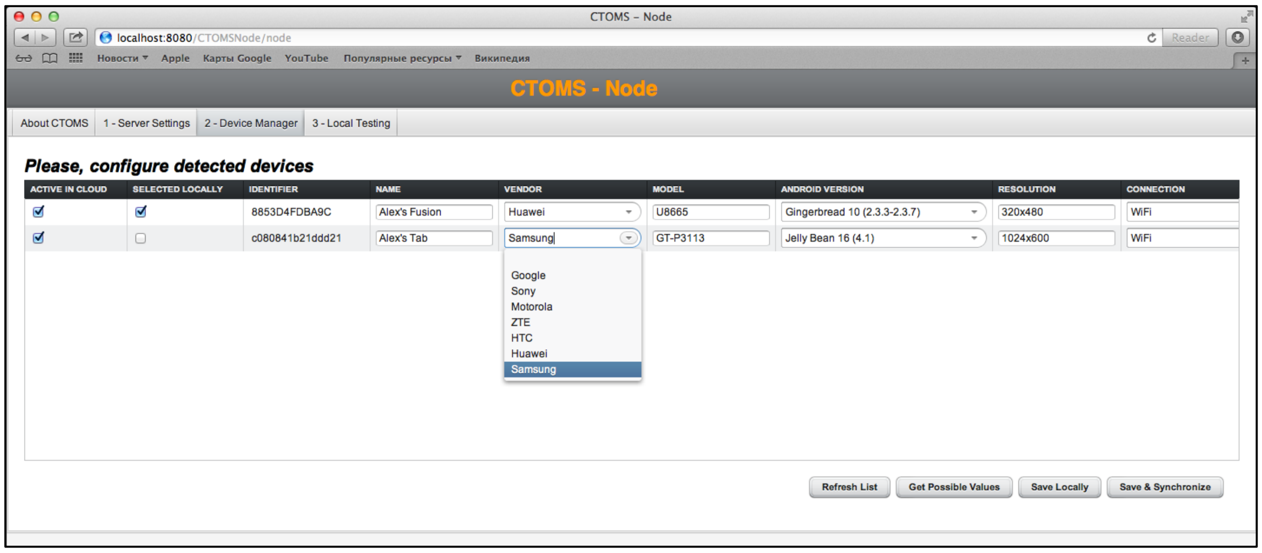
Please, provide necessary settings

Absolute path to ADB util	/Users/user/android-sdks/platform-tools/adb
Absolute path to MaonkeyRunner util	/Users/user/android-sdks/tools/monkeyrunner
URL to Master Server	http://ctoms-strg.appspot.com
Node Name	AlexNode
Node Password	•••
Node URL	https://ecuctoms.fwd.wf/CTOMSNode

Your assigned Node token: 1572

```
You have new mail.
MacBook-Air-user:~ user$ forward 8080
Forwarding port 8080 to https://ecuctoms.fwd.wf
Ctrl-C to stop forwarding
```

Figure 11: The CTOMS Node Screenshot for General Settings



Please, configure detected devices

ACTIVE IN CLOUD	SELECTED LOCALLY	IDENTIFIER	NAME	VENDOR	MODEL	ANDROID VERSION	RESOLUTION	CONNECTION
☑	☑	8853D4FDBA9C	Alex's Fusion	Huawei	U8665	Gingerbread 10 (2.3.3-2.3.7)	320x480	WiFi
☑	☐	c080841b21ddd21	Alex's Tab	Samsung	GT-P3113	Jelly Bean 16 (4.1)	1024x600	WiFi

Figure 12: The CTOMS Node Screenshot for Configuration of Devices

The last suggested configuration step is to make the CTOMSFakeCam Android app be a default camera app on each connected device. This gives the ability to specify images by URL as taken by the camera during testing. Instead of a usual camera tester creating a test script, users should expect a screen with a text input field to enter URL of a certain image and a button to snap a photo, i.e., to use a provided image (see case study number 2). The implementation is similar to [158], but differs with substituting a photo by an image specified by the URL.

Additionally, some Android devices support settings to show touches and screen coordinates for UI debugging. Those can be activated for all connected mobile devices.

All CTOMS Node configurations can be updated remotely (through web access to the application on a node machine), that can be useful especially in case of device settings adjustments. The CTOMS Node also supports local testing to verify the set-up. The following two advices are given to the user of CTOMS about testing:

1. Use a test script similar to that provided in Appendixes A and C of this thesis, i.e., simple Monkeyrunner scripts for a one device with the identical function to take a screenshot. This is needed to insure the correctness of the script update to support simultaneous execution on several devices.
2. If the user needs to find a package and Main Activity of a third-party APK file to provide them in a test script (what to launch), different variants exist, e.g., the following command "aapt dump xmltree AndroidManifest.xml" (aapt is another standard Android SDK tool). As a possible further extension to the framework, it is planned to fully automate via CTOMS interface the process of downloading and installation of an app from Google Play for research testing (user will just have to select which app he wants to test).

The following subsections describe technical solutions of the key functionalities implementation, including general project structure and interactions with Android SDK.

4.3.1 Overall Structure and Deployment

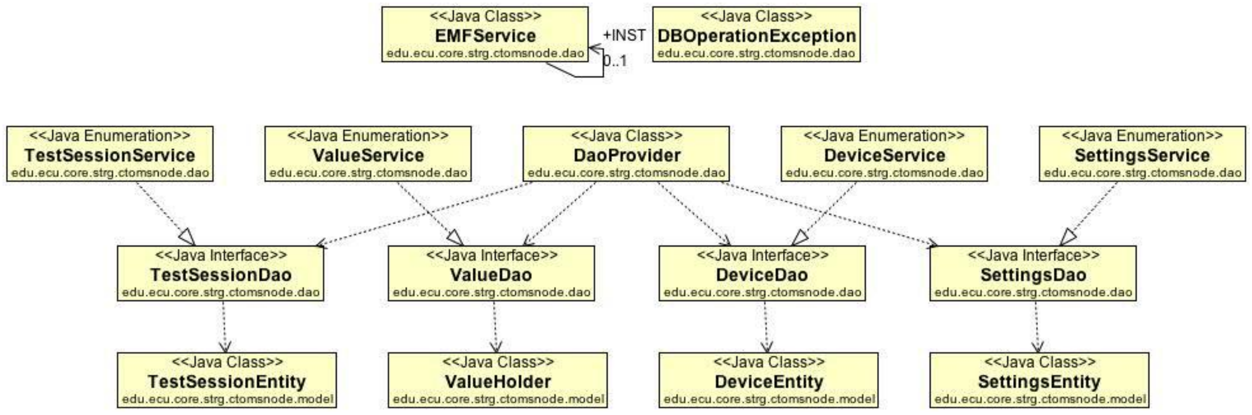
If a user wants to participate as a contributor or hardware/OS developer, he/she needs to download the CTOMS Node web application. It can be installed as any web java application or just by running "java-jar CTOMSNode.war." For convenience, it was packed as a self-executable war file with embedded light Tomcat 7 server (servlet container).

Technical notes on the CTOMS Node implementation are as follows:

- The VAADIN framework [159] was used to implement a whole user web interface because it provides the rapid development of modern web applications (not websites) as desktop ones.
- Internal lightweight Derby database [160] was used to organize the embedded database.
- Jersey JAX-RS reference implementation was used to expose RESTful web services [161].
- The testing queue organizes the sequential execution of the different testing sessions within web server and is implemented using Hazelcast distributed lock [162].
- All test artifacts, including the results, are being stored in the file system with the special hierarchy of directories.

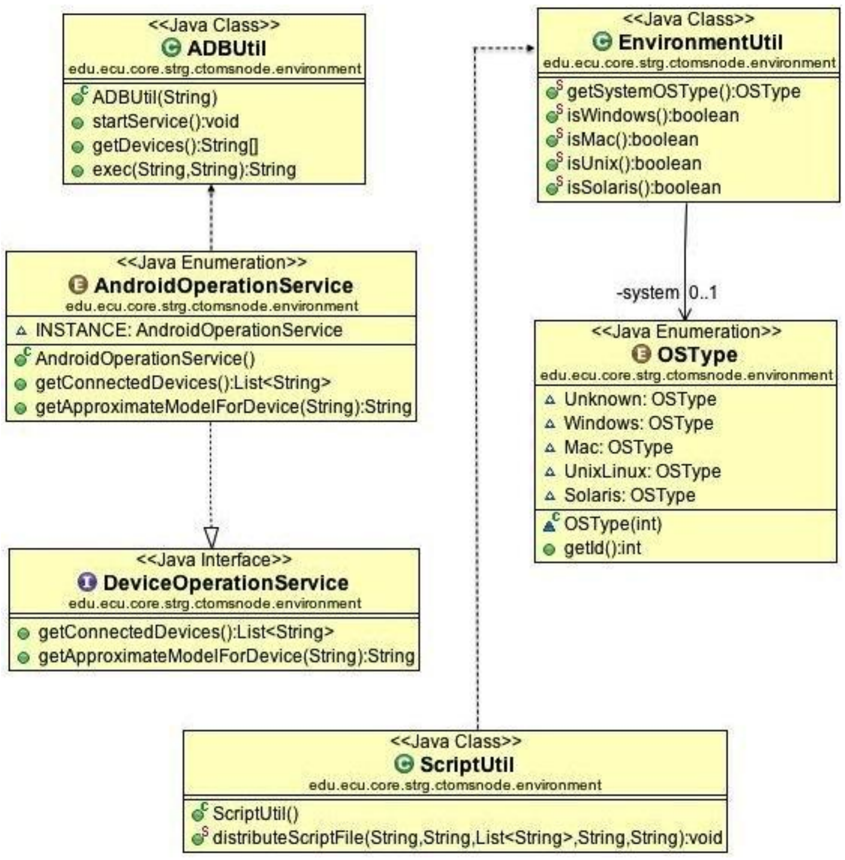
The details of the project structure at the source code level are presented in the following three figures. Figure 13 provides a UML class diagram of the model entities and data access objects (DAO). Work with database is implemented through JPA technology, and special Java classes (mapped to correspondent tables in the database) were created to keep device configurations (DeviceEntity), general settings (SettingsEntity), information about local test sessions (TestsSessionEntity), and approved values for parameters received from master

(ValueHolder). The singleton DaoProvider represents a factory that provides service classes for each entity table. These services classes (like DeviceService or TestSessionService) provide methods to perform needed CRUD operations.



<<Java Class>>
EMFService
edu.ecu.core.strg.ctomsnode.dao
+INST
0..1
<<Java Class>>
DBOperationException
edu.ecu.core.strg.ctomsnode.dao
<<Java Enumeration>>
TestSessionService
edu.ecu.core.strg.ctomsnode.dao
<<Java Enumeration>>
ValueService
edu.ecu.core.strg.ctomsnode.dao
<<Java Class>>
DaoProvider
edu.ecu.core.strg.ctomsnode.dao
<<Java Enumeration>>
DeviceService
edu.ecu.core.strg.ctomsnode.dao
<<Java Enumeration>>
SettingsService
edu.ecu.core.strg.ctomsnode.dao
<<Java Interface>>
TestSessionDao
edu.ecu.core.strg.ctomsnode.dao
<<Java Interface>>
ValueDao
edu.ecu.core.strg.ctomsnode.dao
<<Java Interface>>
DeviceDao
edu.ecu.core.strg.ctomsnode.dao
<<Java Interface>>
SettingsDao
edu.ecu.core.strg.ctomsnode.dao
<<Java Class>>
TestSessionEntity
edu.ecu.core.strg.ctomsnode.model
<<Java Class>>
ValueHolder
edu.ecu.core.strg.ctomsnode.model
<<Java Class>>
DeviceEntity
edu.ecu.core.strg.ctomsnode.model
<<Java Class>>
SettingsEntity
edu.ecu.core.strg.ctomsnode.model

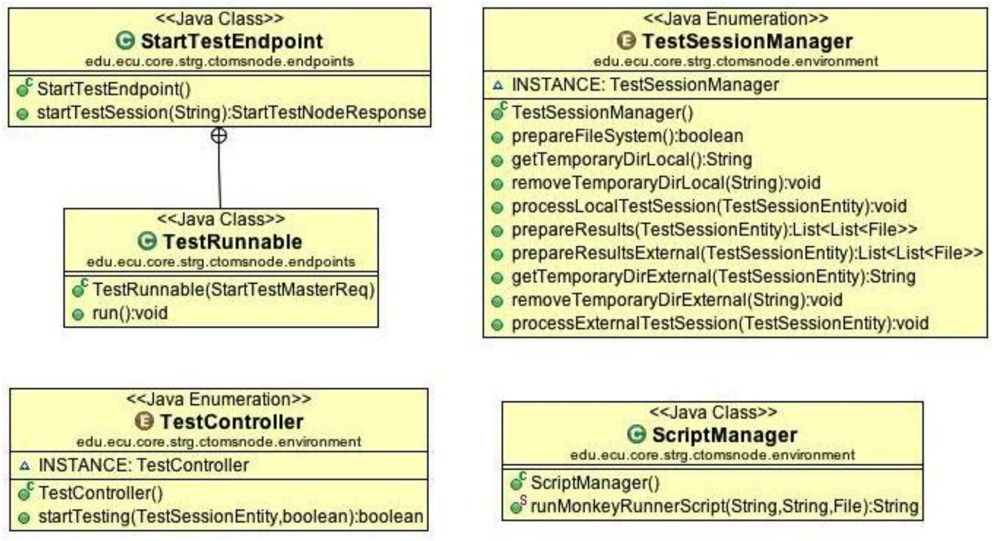
Figure 13: The CTOMS Node Class Diagram for the Data Level



<<Java Class>>
ADBUtil
edu.ecu.core.strg.ctomsnode.environment
ADBUtil(String)
startService():void
getDevices():String[]
exec(String,String):String
<<Java Class>>
EnvironmentUtil
edu.ecu.core.strg.ctomsnode.environment
getSystemOSType():OSType
isWindows():boolean
isMac():boolean
isUnix():boolean
isSolaris():boolean
-system 0..1
<<Java Enumeration>>
AndroidOperationService
edu.ecu.core.strg.ctomsnode.environment
INSTANCE: AndroidOperationService
AndroidOperationService()
getConnectedDevices():List<String>
getApproximateModelForDevice(String):String
<<Java Enumeration>>
OSType
edu.ecu.core.strg.ctomsnode.environment
Unknown: OSType
Windows: OSType
Mac: OSType
UnixLinux: OSType
Solaris: OSType
OSType(int)
getId():int
<<Java Interface>>
DeviceOperationService
edu.ecu.core.strg.ctomsnode.environment
getConnectedDevices():List<String>
getApproximateModelForDevice(String):String
<<Java Class>>
ScriptUtil
edu.ecu.core.strg.ctomsnode.environment
ScriptUtil()
distributeScriptFile(String,String,List<String>,String,String):void

Figure 14: The CTOMS Node Class Diagram for Working with Android SDK

Figure 14 illustrates a set of utility classes to work with the environment, specifically, Android SDK. For instance, ADBUtil and AndroidOperationService provide an interface to retrieve a list of connected devices and their models (getApproximateModelForDevice method, approximate because this approach uses parsing of a device's properties retrieved from the Android shell and the format of information is not strictly standardized). ScriptUtil processes the initial test script by replacing a code for one connected device with the code that operates selected devices (using their tokens). The same screenshots are being taken as is specified in the original script, but with updated script they are taken for each device, sorted by a timestamp, and put in separate dedicated folders. Further screenshots can be easily retrieved and shown in an organized way. More details about working with Android SDK are discussed in the next subsection.



<<Java Class>>
StartTestEndpoint
edu.ecu.core.strg.ctomsnode.endpoints
StartTestEndpoint()
startTestSession(String):StartTestNodeResponse
<<Java Class>>
TestRunnable
edu.ecu.core.strg.ctomsnode.endpoints
TestRunnable(StartTestMasterReq)
run():void
<<Java Enumeration>>
TestSessionManager
edu.ecu.core.strg.ctomsnode.environment
INSTANCE: TestSessionManager
TestSessionManager()
prepareFileSystem():boolean
getTemporaryDirLocal():String
removeTemporaryDirLocal(String):void
processLocalTestSession(TestSessionEntity):void
prepareResults(TestSessionEntity):List<List<File>>
prepareResultsExternal(TestSessionEntity):List<List<File>>
getTemporaryDirExternal(TestSessionEntity):String
removeTemporaryDirExternal(String):void
processExternalTestSession(TestSessionEntity):void
<<Java Enumeration>>
TestController
edu.ecu.core.strg.ctomsnode.environment
INSTANCE: TestController
TestController()
startTesting(TestSessionEntity,boolean):boolean
<<Java Class>>
ScriptManager
edu.ecu.core.strg.ctomsnode.environment
ScriptManager()
runMonkeyRunnerScript(String,String,File):String

Figure 15: The CTOMS Node Class Diagram for Performing Tests

Figure 15 shows the classes that perform the business logic of testing. TestSessionManager manages test sessions and required work with the file system. TestController starts the testing process by finally using ScriptManager (driver class to run Monkeyrunner scripts). StartTestEndpoint is a REST service that the master application invokes to submit a test task. TestRunnable is presented to highlight how the waiting queue of a test session is organized just by locking threads (using a distributed lock that works across processes of a web server).

All class diagrams were constructed using a reverse engineering technique to check the correctness of the implemented ideas.

The common approach used was the organization of the code like a set of singleton managers (implemented as a numeration in Java) powered by correspondent interfaces. This decision is justified by the fact that the provided functionality is greatly heterogeneous, and so keeping logic classes independent will increase the ability to reuse or change. Additionally, further extensions like new drivers will appear like new manager classes supporting similar interfaces and combined by a factory provider.

4.3.2 Interactions with Android SDK

The current version of the node application uses the "adb" command line tool (ADB, Android Debug Bridge) from SDK to retrieve information about connected devices and the Monkeyrunner tool for testing. Test scripts are assumed to be real test scripts used by a development team on its own site, but aimed at screenshot-based testing. This means the script should use standard Monkeyrunner facilities to save screenshots for desired checkpoints and that the user-tester will have the ability to compare screenshots taken from different models.

The stable and portable version of operating with ADB and Monkeyrunner required some non-trivial solutions and workarounds. First, problems occur with a single format of tools execution from Java that will work on both Windows and Linux-related systems (CTOMS Node was fully tested on Mac OS). The problem with Monkeyrunner was the most representative. It was caused by the fact that Monkeyrunner is an executable script program in Android SDK for

Mac OS, but is a bat-file on Windows that cannot be launched as an executable file. This forced us to use EnvironmentUtil (Figure 14) to detect the current OS.

Second, the advanced engine to work with external tools was coded so that it (1) works with different paths, (2) handles output streams in a separate threads (to not miss or lock anything), (3) properly closes all streams in any case, (4) reliably handles exceptions, and (5) reads output to determine result of testing (success or some error message that should be shown as a status for a test session).

All this code is rather self-documented and can be usefully reused if published as open source (or distributed in other way).

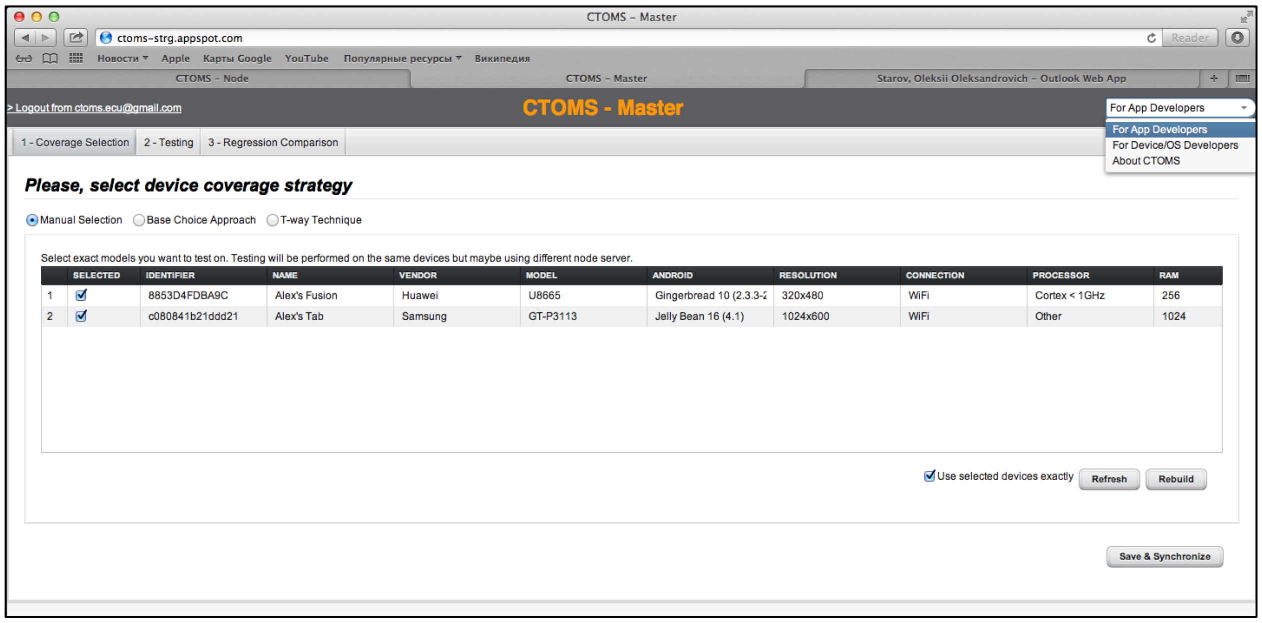
4.4 The CTOMS Master Implementation

The master application provides the user with the list of all devices connected to the system and organizes the testing of the same application on different nodes, if needed. An initial load balancing approach is used, i.e., the selection of the freest nodes that have the desired model connected (see next subsection).

The master application adds additional functionality for mobile app developers to apply combinatorial techniques for testing. The user can chose a set of models to test on, not manually, but using, for instance, a pair-wise approach. An example of this application for different Android configurations can be found at [21]. To calculate the required test cases, CTOMS uses the ACTS tool provided by NIST and integrated as a library within the project.

It was decided that two possible perspectives ("For App Developers" and "For Device/OS Developers") are enough to cover all demanded use cases and additional scenarios of multidirectional testing.

The *For App Developers* perspective is mainly focused on testing a new app (that is the most common use case). Additionally, a user can test an APK of an existing application, choose a set of devices with fixed hardware or OS to perform investigations, and execute a test script without APK (testing of OS). All publicly available devices (from all nodes) are accessible through this perspective.


CTOMS – Master

ctoms-strg.appspot.com

Новости Apple Карты Google YouTube Популярные ресурсы Википедия

CTOMS – Node | CTOMS – Master | Starov, Oleksii Oleksandrovich – Outlook Web App

> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For App Developers / For App Developers / For Device/OS Developers / About CTOMS

1 - Coverage Selection | 2 - Testing | 3 - Regression Comparison

Please, select device coverage strategy

[✓ Manual Selection] Base Choice Approach T-way Technique

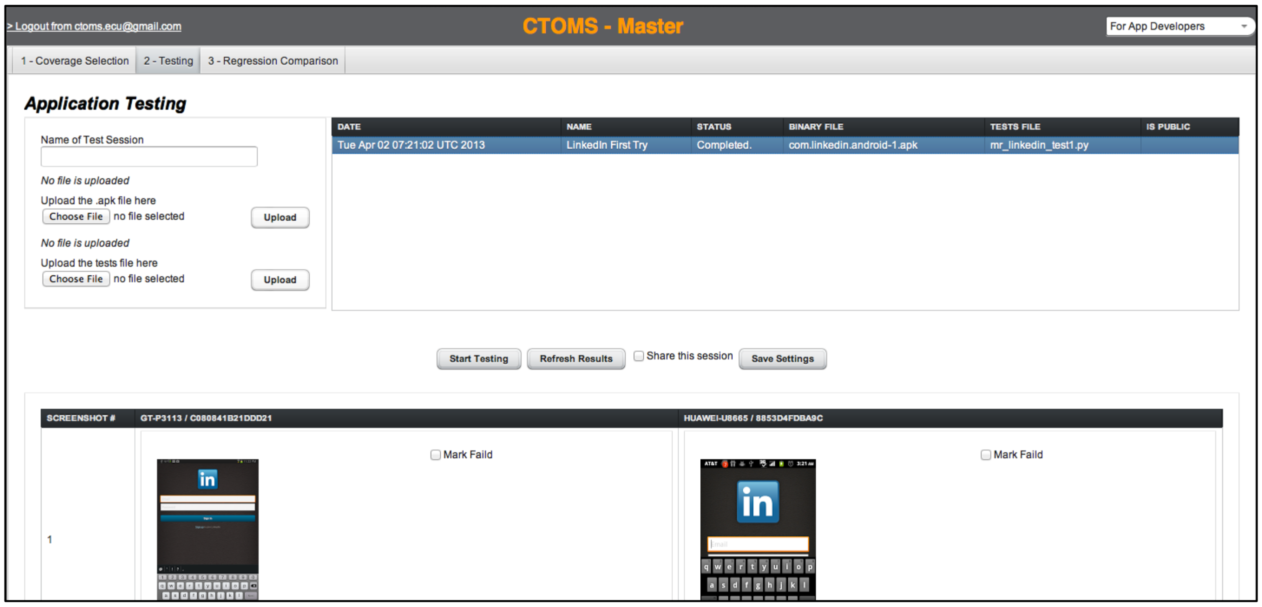
Select exact models you want to test on. Testing will be performed on the same devices but maybe using different node server.

	SELECTED	IDENTIFIER	NAME	VENDOR	MODEL	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM
1	✓	8853D4FDBA9C	Alex's Fusion	Huawei	U8665	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256
2	✓	c080841b21ddd21	Alex's Tab	Samsung	GT-P3113	Jelly Bean 16 (4.1)	1024x600	WiFi	Other	1024

✓ Use selected devices exactly | Refresh | Rebuild

Save & Synchronize

Figure 16: The CTOMS Master Screenshot for Device Selection


> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For App Developers

1 - Coverage Selection | 2 - Testing | 3 - Regression Comparison

Application Testing

Name of Test Session

No file is uploaded

Upload the .apk file here

Choose File no file selected Upload

No file is uploaded

Upload the tests file here

Choose File no file selected Upload

DATE	NAME	STATUS	BINARY FILE	TESTS FILE	IS PUBLIC
Tue Apr 02 07:21:02 UTC 2013	LinkedIn First Try	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	

Start Testing | Refresh Results | Share this session | Save Settings

SCREENSHOT #	GT-P3113 / C080841B21DDD21	HUAWEI-U8665 / 8853D4FDBA9C
1	Mark Faild	Mark Faild

Figure 17: The CTOMS Master Screenshot for Testing Tab

The *For Device/OS Developers* perspective is mainly focused on testing a new device/OS. A user can test the own device connected to the cloud against those available in the system test logs of mobile apps (or just test scripts), and whether the device represents new hardware or contains an installed new OS version. Additionally, the user can choose a set of devices with fixed hardware or OS to perform investigations. All connected devices in the system can be found in the list.

Figure 16 presents a screenshot of the perspective for app developers on the CTOMS Master, or more precisely, the screen for the manual exact selection of devices to test an app on. More screenshots, including explanations of the combinatorial techniques interface, can be found in Chapter 5. Figure 17 illustrates the testing tab. The screen contains a menu to start a new test session and upload correspondent artifacts. A series of screenshots are shown for the LinkedIn app testing on a pair of devices.

4.4.1 Google App Engine Deployment

The master is implemented as a Google App Engine Java web application. A user interface is also developed using the popular VAADIN framework, and the RESTful web services tier uses Jersey implementation. File storing is implemented through GAE Blobstore, and the usage of cloud data storage is considered as an alternative variant for future versions [150].

Most screens of the VAADIN interface are table-based (both for CTOMS Master and CTOMS Node). The inheritance of the BeanItemContainer is widely used. There are some acknowledged issues with VAADIN and Google App Engine compatibility, but all necessary guidelines were followed [155].

The source code of the project consists of the following parts: entities and work with GAE database, general managers, test controls, RESTful web services, and test technique providers.

The model is similar but much greater than in the CTOMS Node. Such entities like test session for device testing, test result record (per device), test settings, and approved values of device parameters are added. Test settings are used to immediately maintain entered settings by a user, so to not lose them during screen transition. At the same time, the CTOMS Master operates a lot with DevicePattern class that can be derived from DeviceEntity.

Figure 18 shows examples of used managers to track the current user, to operate with information about connected nodes, to log errors, etc. It also shows the main classes for test launching. TestDistributor selects the nodes based on requested devices or patterns (parameter configurations) with respect to load balancing (see next subsection). NodeRestClient uses Jersey Client to invoke ping rest service (to start testing) on a node.

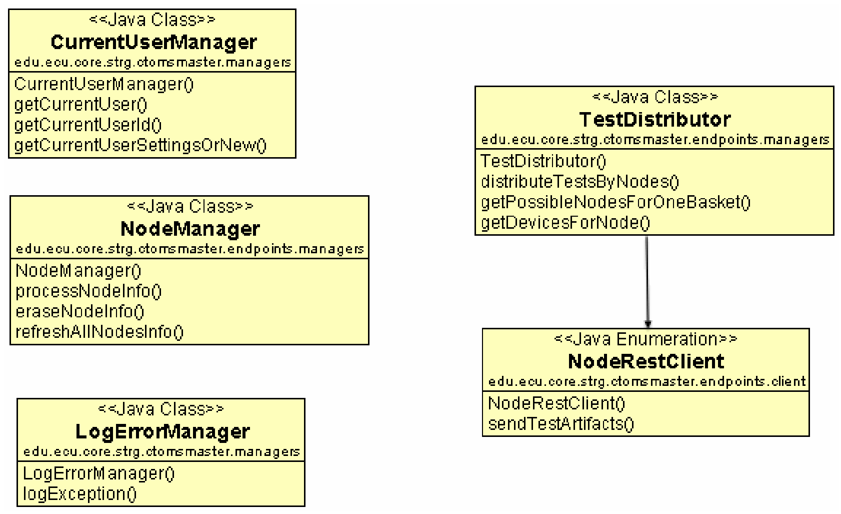


Figure 18: The CTOMS Master Class Diagram for Managers and Test Controls

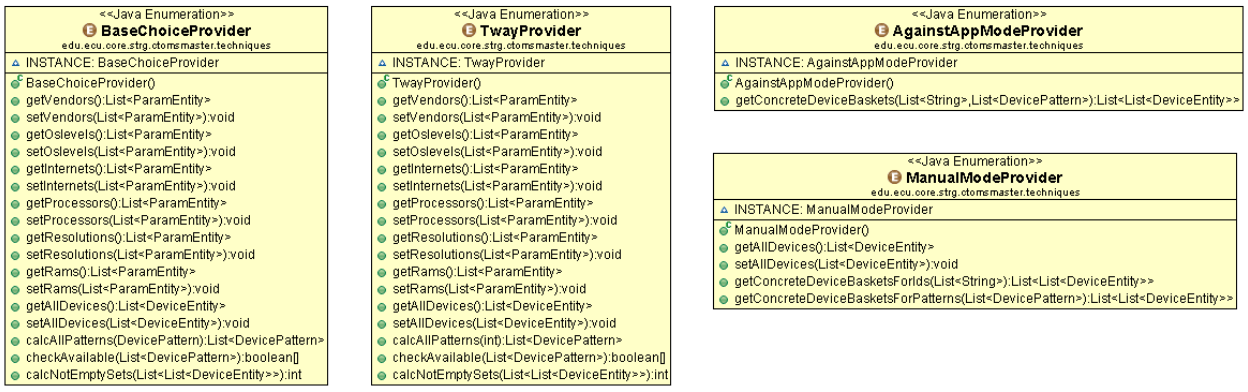

<<Java Enumeration>>
BaseChoiceProvider
edu.ecu.core.strg.ctomsmaster.techniques
INSTANCE: BaseChoiceProvider
BaseChoiceProvider()
getVendors():List<ParamEntity>
setVendors(List<ParamEntity>):void
getOslevels():List<ParamEntity>
setOslevels(List<ParamEntity>):void
getInternets():List<ParamEntity>
setInternets(List<ParamEntity>):void
getProcessors():List<ParamEntity>
setProcessors(List<ParamEntity>):void
getResolutions():List<ParamEntity>
setResolutions(List<ParamEntity>):void
getRams():List<ParamEntity>
setRams(List<ParamEntity>):void
getAllDevices():List<DeviceEntity>
setAllDevices(List<DeviceEntity>):void
calcAllPatterns(DevicePattern):List<DevicePattern>
checkAvailable(List<DevicePattern>):boolean[]
calcNotEmptySets(List<List<DeviceEntity>>):int
<<Java Enumeration>>
TwayProvider
edu.ecu.core.strg.ctomsmaster.techniques
INSTANCE: TwayProvider
TwayProvider()
getVendors():List<ParamEntity>
setVendors(List<ParamEntity>):void
getOslevels():List<ParamEntity>
setOslevels(List<ParamEntity>):void
getInternets():List<ParamEntity>
setInternets(List<ParamEntity>):void
getProcessors():List<ParamEntity>
setProcessors(List<ParamEntity>):void
getResolutions():List<ParamEntity>
setResolutions(List<ParamEntity>):void
getRams():List<ParamEntity>
setRams(List<ParamEntity>):void
getAllDevices():List<DeviceEntity>
setAllDevices(List<DeviceEntity>):void
calcAllPatterns(int):List<DevicePattern>
checkAvailable(List<DevicePattern>):boolean[]
calcNotEmptySets(List<List<DeviceEntity>>):int
<<Java Enumeration>>
AgainstAppModeProvider
edu.ecu.core.strg.ctomsmaster.techniques
INSTANCE: AgainstAppModeProvider
AgainstAppModeProvider()
getConcreteDeviceBaskets(List<String>,List<DevicePattern>):List<List<DeviceEntity>>
<<Java Enumeration>>
ManualModeProvider
edu.ecu.core.strg.ctomsmaster.techniques
INSTANCE: ManualModeProvider
ManualModeProvider()
getAllDevices():List<DeviceEntity>
setAllDevices(List<DeviceEntity>):void
getConcreteDeviceBasketsForIds(List<String>):List<List<DeviceEntity>>
getConcreteDeviceBasketsForPatterns(List<DevicePattern>):List<List<DeviceEntity>>

Figure 19: The CTOMS Master Class Diagram for Test Techniques Providers

4.4.2 Load-balancing for Test Distribution

In the current CTOMS architecture, load-balancing mechanisms for test distribution are fully implemented by the master. This thesis deals with a simplified version of the generally complex optimization problem. The CTOMS Master operates with the size of test queues per a single node, while in the full version it should consider the loading of a separate device. The justification for this approach is the fact that in this CTOMS version (using Monkeyrunner) test sessions on each node are being executed sequentially, i.e., one at a time, so there is no need to consider tests for different devices separately.

Let's define the actual problem of test distribution. After a user manually selects a list of concrete device models to test on or a list of device patterns (set of parameters) is generated according to some combinatorial coverage, the system analyzes all connected smartphones and generates the list of device baskets. Each basket contains "equal" devices for the current session (i.e., identical models in the case of manual selections or devices that correspond to certain patterns of parameters). To fulfill the user's request, the CTOMS Master should then select one device from each basket on which to perform the testing. This selection must be made in optimal way to provide the fastest testing. Figure 20 illustrates the process. The simple lines show possible selections, the bold lines show selected concrete devices, and the dashed line show scenario if we consider load per a device.

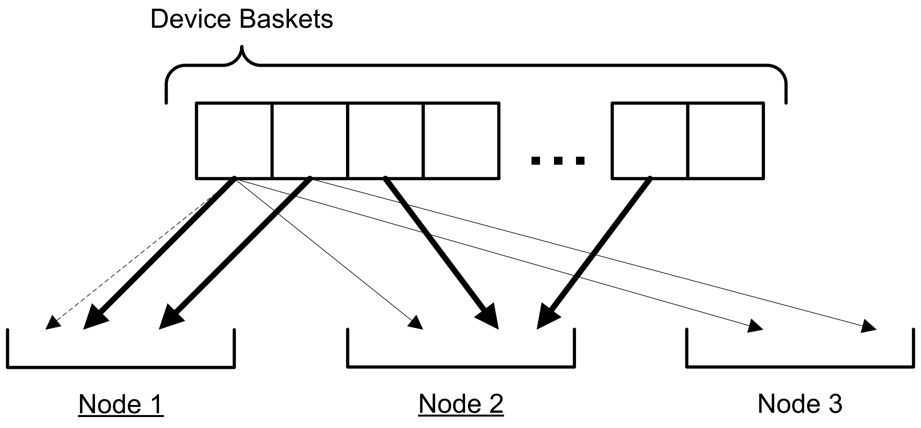


Figure 20: The Test Distribution Process

To make the selection of concrete devices effective, the CTOMS Master should consider the current load (i.e., number of test sessions in the queue) of each node. In the current study, we do not consider time costs for task sending, so there is not much benefit if we try to minimize the number of nodes. At the same time, the results are uploaded to the master per each device. So the following summarizes the problem and general simplified solution that can be implemented.

Input: list of sets of devices connected to the system

Output: list of nodes and for each node a set of devices to launch a test case on

Solution: The algorithm should select such a list of nodes *l*, that *max(load(l_i))* will be minimal (they will work in parallel), where *load()* can return size of the current test queue on a certain node.

The implementation of this algorithm requires the CTOMS Node to report its current load when sending node info to the master (0 during the "registration" and changed value after "ping" and finish of testing).

The current version of CTOMS does not implement the aforementioned approach. The reason is that, given the current small sizes of the platform usage, the current efficiency is enough. It will be sensible to spend time and effort on advanced test distribution implementation with start of crowdsourcing process. Currently, a greedy algorithm is used that goes through the list of device baskets, takes an arbitrary node that contains a representative of the basket, saves it to results, and figures out what other baskets can be covered by this node and be removed from the list and further consideration. The greedy approach can be instantly improved by choosing not an arbitrary node, but a node that covers the bigger number of baskets (it can be gained by sorting the lists of possible nodes for each basket).

4.5 Master-Node Communication

This section explains the format and process of communication between master and node applications. As was mentioned previously, the constraint was imposed to minimize the number of calls because free usage of Google App Engine has a quote on the number of requests. At the same time, the system should still be easily scalable and convenient in use. All these resulted in the rejection of the interaction pattern with periodical calls for information updates and in extensive use of updates per demand (refresh buttons in UI, update of a node's info only in case of testing request, etc.). However, the architecture of web services was developed with the respect to possible further enabling of automatic data updates. Figure 21 illustrates the main requests and scenario of call sequence. The main points are summarized in the following list:

1. One service and format of the message is used to provide registration and a node's information update (node name, list of devices, passphrase to use private devices, etc.). The registration of a node is secured with a token that should be received from the CTOMS administrator.

2. The master uses a ping service to update a node’s information by demand (or just to check if it is still online). At the same time, the ping is used to invoke a test session. The node immediately answers with the current status and resumes testing in asynchronous mode. The waiting queue to process test session sequentially is organized implicitly by using Hazelcast distributed lock [162].
3. As a consequence to the previous item, CTOMS Node should be accessible by HTTP from the master (public IP, ssh tunneling like [151] and [152], etc.). This is the main disadvantage of absence of periodical requests. Otherwise, a node could sent ping requests to the master by timer with small interval and quickly receive actions in the response.
4. The CTOMS Node downloads test artifacts (APK binary file and Python test script) not via the direct call to the GAE Data Store (files are not publicly exposed), but through the request to master. It gives an opportunity to provide access secured by a password to the files (actually, every call can be secured this way, and HTTPS is also possible on GAE). No caching of test artifacts on a node’s site is currently implemented.
5. Uploading of the results is done per device (to minimize the size of the package). The series of screenshots is sent in the form of multipart data. Another benefit is that user can view partial results before all are uploaded (of course, by pressing a refresh button, because automatic UI updates are disabled for the current reason with the goal of economy).
6. The Figure 21 doesn’t show the calls to get the list of possible parameters and values of a device approved in the master. It is a simple service that just returns the whole array of data.

7. Figure 21 does not show the calls to Data Base, and CTOMS Master uses it each time the data update is requested.

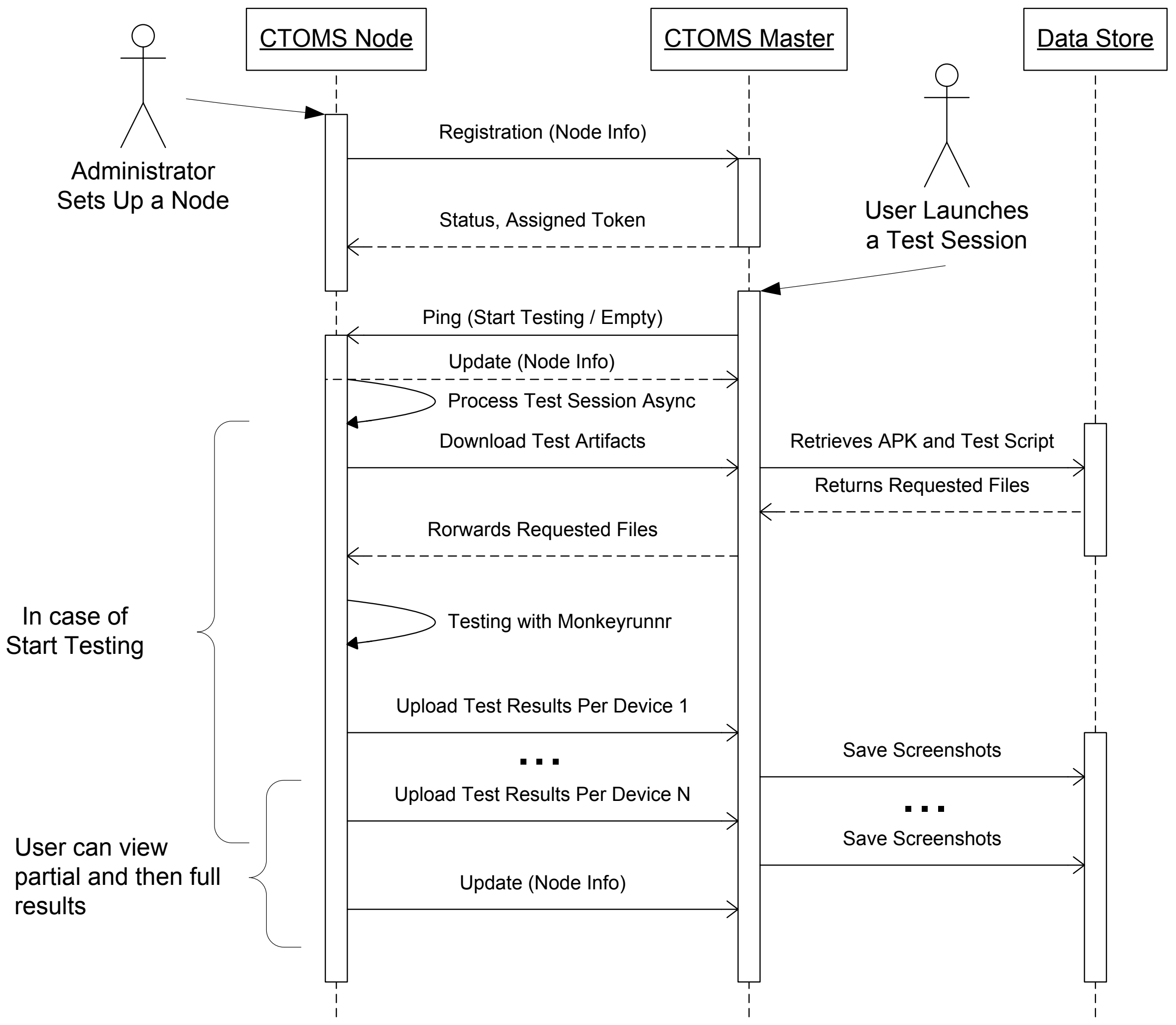


Figure 21: UML Sequence Diagram of Master-Node Interactions

Jersey library [161] was used to implement RESTful web services. The following are examples of JSON messages for node registration and start testing ping, respectively. JSON format was used because of its efficiency and popularity among mobile developers. Furthermore, the CTOMS platform can be extended with correspondent documented APIs. The possible efficiency enhancement is the leveraging of Protocol Buffers [163].

An example of the message to register a node:

```
{
  "nodeInfo": {
    "devices": [
      {
        "active": "true",
        "id": "1",
        "internet": "WiFi",
        "locale": "en",
        "model": "U8665",
        "name": "Alex's Fusion",
        "openPublicly": "true",
        "oslevel": "Gingerbread 10 (2.3.3-2.3.7)",
        "processor": "Cortex < 1GHz",
        "ram": "256",
        "resolution": "320x480",
        "selected": "true",
        "supportAVC": "false",
        "token": "8853D4FDBA9C",
        "vendor": "Huawei"
      },
      {
        "active": "true",
        "id": "4",
        "internet": "WiFi",
        "locale": "EN",
        "model": "GT-P3113",
        "name": "Alex's Tab",
        "openPublicly": "true",
        "oslevel": "Jelly Bean 16 (4.1)",
        "processor": "Other",
        "ram": "1024",
        "resolution": "1024x600",
        "selected": "false",
        "supportAVC": "false",
        "token": "c080841b21ddd21",
        "vendor": "Samsung"
      }
    ],
    "name": "AlexNode",
    "nodeUrl": "https:\/\/ecuctoms.fwd.wf\/CTOMSNode",
    "passphrase": "456",
    "queue": "0",
    "token": "63001"
  }
}
```

An example of the ping message from CTOMS Master:

```
{
  "againstApp": "false",
  "apkName": "com.blackboard.android.central.ecu-1.apk",
  "apkUrl": "download\/securedBlobDownloader?blob-key=some_key",
  "deviceTokens": [
    "8853D4FDBA9C",
    "c080841b21ddd21"
  ],
  "nodeToken": "70001",
  "scriptUrl": "download\/securedBlobDownloader?blob-key=some_key",
  "sessionToken": "63002"
}
```

The current minimalistic web services tier is light and powerful enough to support all scaling needs. Other testing directions like a device against app does not require additional services because of the abstraction—the same testing is performed on a selected new device, and then the CTOMS Master compares the screenshots with available nominal ones.

4.6 Notes on Scalability and Universality

It was previously mentioned several times that CTOMS has a scalable and universal architecture. This short section explains what this means and how it is implemented.

Scalability means that we easily can add nodes to the platform. It does not require a lot of efforts from a contributor and configuration of the CTOMS Node is quick. Lightweight services provide enough functionality to connect the whole system, and the CTOMS Master supports the work with multiple nodes. The only current limitation is the requirement that the node must be accessible through HTTP calls from the master, but it is explained by economic reasons, and a solution of continuous periodical status requests from node is proposed for further versions.

Correct administration of connected Android devices is fully the task of the participant's site in the current version. Some automatic approaches to analyze connected devices and retrieve all needed settings can be added later. The same situation with restrictions to uploaded test scripts and their check.

Universal architecture in this CTOMS version means possible reuse of implemented solutions and design for the following purposes:

- To support other tests automation drivers [117, 74]
- To support other mobile platforms [19]
- To integrate and add other testing techniques or additional services

As the third item was comprehensively discussed above, the first two address the similarities of working with different test automation tools or the similarities of mobile platforms (like SDK tools, debugging of smartphones, etc.).

Of course, reuse is currently possible still on the code level, i.e., some changes are needed, but minimal ones. For instance, to support other test automation tools like Robotium [117], the implementation of a special driver class is needed, and some extensions to the user interface (ability to select a test script type), but the main architecture of the system will remain. Moreover, this main architecture can be used to implement a CTOMS addition for Windows Phone or another mobile platform. The process of interactions with correspondent SDK, retrieving a list of connected devices, sending test scripts, launching testing—will be similar. Of course, some differences exist and special workarounds will be needed, but key ideas will still remain. Such issues can occur with different platforms' restrictions, but in the case of open-source mobile systems, it should not be a problem because a solution (at least a complicated one) will always exist.

In the conclusion to this chapter, it is sensible to define the limitations of the current CTOMS implementations and its solutions described above. From the expert's point of view and based on the conducted case studies discussed in the next chapter, the platform will work with satisfactory performance and convenience in case of the following:

- *Each node provides approximately up to 10 connected devices.*

- *Test scripts satisfy a simple pattern (see case studies in Chapter 4) and are aimed at taking 10–20 screenshots (per device). This means, usage of medium test cases, i.e., check of a one functionality per scrip, is preferable.*
- *Support of more complicated test scripts (that use some extensions) may require improvements of the script parallelization logic. The issue can be mitigated in further versions by using more advanced test automation tools.*
- *This CTOMS version is limited to provide testing through screenshot comparison (both UI and functional) that can be enhanced with pass/fail reports and device (crash) logs.*
- *The platform can contain dozens of nodes (i.e. crowdsourcing participants and contributors), but launching a test on more than 10 devices can be lengthy by time. Ability to view partial results (for some devices) helps, but during long tests the probability of a connection lost from a node grows.*
- *The CTOMS Node was tested on its compatibility with Mac OS and Windows.*
- *Periodical problems with VAADIN and GAE compatibility (like session management) can occur.*
- *Test script adjustments and re-testing are periodically needed because of such things like different Internet speeds on different devices, etc. Obviously, following up manual testing is also necessary.*
- *The combinatorial coverage calculations implemented in CTOMS and generally testing on a range of devices are useful in case of a rather stable build of a mobile application or for experiments with an app from production. During initial stages of the development, remote access to any one smartphone is typically enough.*

The list above provides empirical evaluation of CTOMS, but with the growth of a crowdsourcing system (i.e., involving participants), and hence a real geographically distributed system, more systematic analysis and assessment will be needed.

There is a room for technical improvements and enhancements of CTOMS. There is an idea to make the code open source, i.e., to leverage crowdsourcing benefits for implementation improvements and bug fixing as well. It also will open the ability to reuse used architecture and solutions, or even to use CTOMS as an internal test automation system, i.e., privately deployed cloud within mobile development companies.

CHAPTER 5: CASE STUDY

This chapter describes the two first-case studies of using the CTOMS. They serve as a demonstration of demanded functionality and novel use-cases and provide a summary of the framework's testing. All case studies described below were conducted using two nodes (Mac and Windows laptops) with connections to three different devices. Proper test distribution and convenient enough efficiency were simultaneously checked. The test scripts were prepared manually beforehand based on the aimed functionality to be verified. During the development of CTOMS, the specially implemented separate small Android application was used for testing and debugging.

5.1 Testing ECU Mobile for Android

The first mobile application tested was the ECU Mobile app for Android. It was chosen as an example of the application created by a rather small development team. This case study had the following aims:

- To demonstrate mobile app testing using CTOMS
- To demonstrate the selection of devices according to a provided coverage
- To evaluate bug detectability of the framework
- To test a hybrid mobile application (native and a web mobile part)

The version of the ECU Mobile under the test was installed from Google Play on March 28, 2013. The APK-file was simply copied from the file system of a mobile device. The name of the package and a start activity (screen) to be launched from the test script were retrieved from the binary using the following command: “aapt dump xmltree AndroidManifest.xml” (aapt is a standard Android SDK tool).

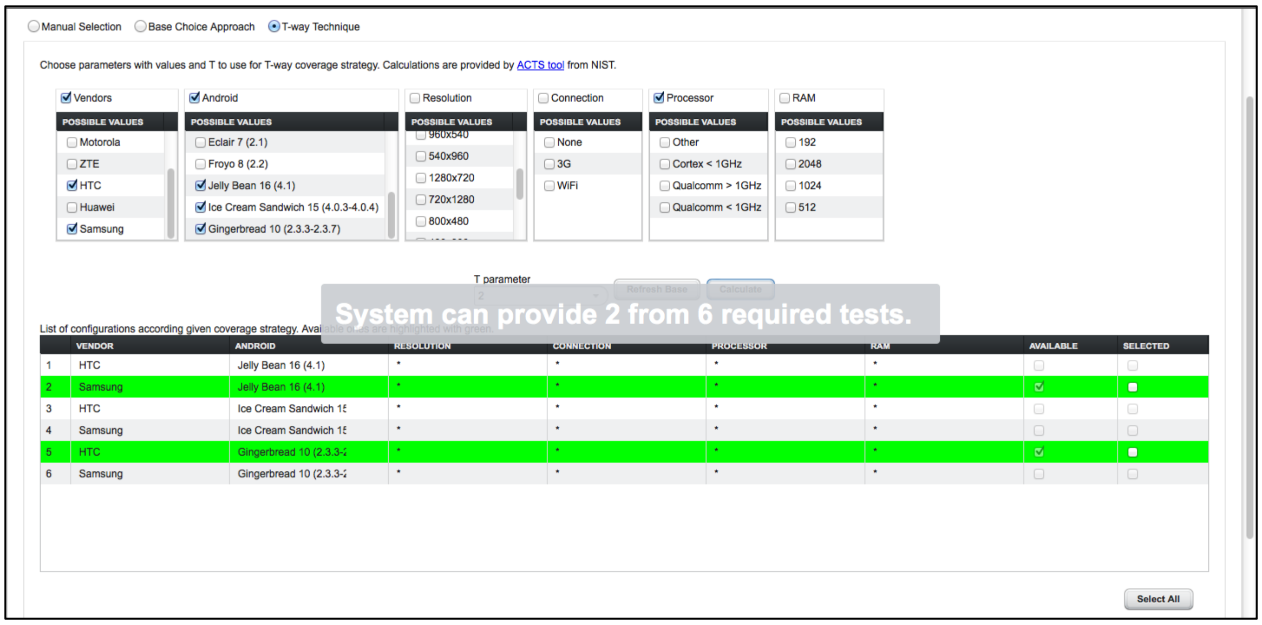
The original Monkeyrunner test script using during case study is provided in Appendix A. The complete screenshot report according to the steps of testing can be examined in using Appendix B.

5.1.1 Main Native App Testing

Though ECU Mobile was tested using the one test script, we can split it into two parts: interactions with a native part of the application and testing of the Inner Pirate Network web app (launched from ECU Mobile and running in the mobile browser).

The following checkpoints (i.e., screenshots to be taken) are described in the test script for the native part:

1. Check appearance of a splash screen.
2. Check appearance of a main screen.
3. Slide to the second screen.
4. Launch Inner Pirate Network by pressing a button.



Manual Selection Base Choice Approach [✓ T-way Technique]

Choose parameters with values and T to use for T-way coverage strategy. Calculations are provided by ACTS tool from NIST.

✓ Vendors	✓ Android	Resolution	Connection	✓ Processor	RAM
POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES
Motorola	Eclair 7 (2.1)	960x540	None	Other	192
ZTE	Froyo 8 (2.2)	540x960	3G	Cortex < 1GHz	2048
✓ HTC	✓ Jelly Bean 16 (4.1)	1280x720	WiFi	Qualcomm > 1GHz	1024
Huawei	✓ Ice Cream Sandwich 15 (4.0.3-4.0.4)	720x1280		Qualcomm < 1GHz	512
✓ Samsung	✓ Gingerbread 10 (2.3.3-2.3.7)	800x480			

T parameter
2

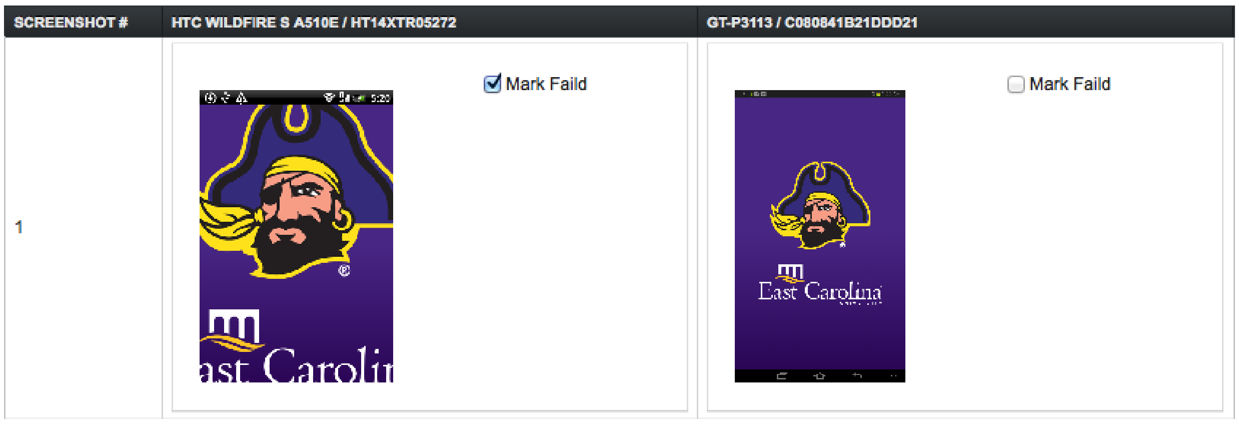
System can provide 2 from 6 required tests.

List of configurations according given coverage strategy. Available ones are highlighted with green.

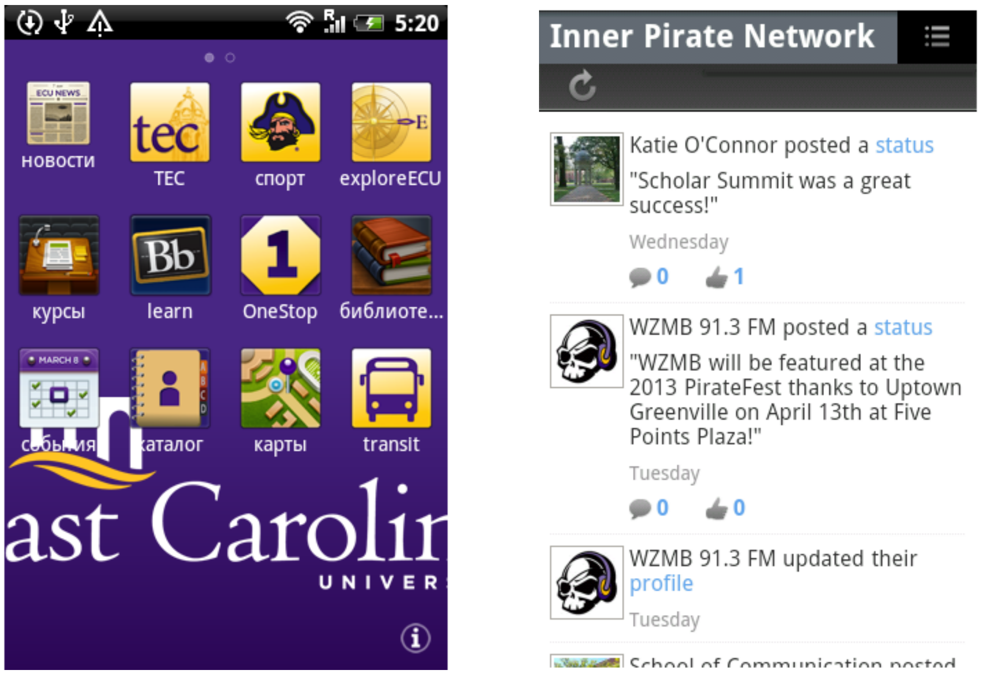
	VENDOR	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM	AVAILABLE	SELECTED
1	HTC	Jelly Bean 16 (4.1)	*	*	*	*		
2	Samsung	Jelly Bean 16 (4.1)	*	*	*	*	✓	
3	HTC	Ice Cream Sandwich 15	*	*	*	*		
4	Samsung	Ice Cream Sandwich 15	*	*	*	*		
5	HTC	Gingerbread 10 (2.3.3-2	*	*	*	*	✓	
6	Samsung	Gingerbread 10 (2.3.3-2	*	*	*	*		

Figure 22: The Screenshot of T-way Coverage

Testing was conducted on a Samsung Galaxy Tab 2 7.0 and an HTC Wildfire S (each on a separate node, Mac one and Windows one, respectively). They were selected from the results of t-way coverage calculation. Figure 22 demonstrates a list of configurations according to the chosen combinatorial technique and available models.



SCREENSHOT #	HTC WILDFIRE S A510E / HT14XTR05272	GT-P3113 / C080841B21DDD21
1	☑ Mark Faild	☐ Mark Faild



Katie O'Connor posted a status
"Scholar Summit was a great success!"
Wednesday
0 1

WZMB 91.3 FM posted a status
"WZMB will be featured at the 2013 PirateFest thanks to Uptown Greenville on April 13th at Five Points Plaza!"
Tuesday
0 0

WZMB 91.3 FM updated their profile
Tuesday

School of Communication posted

Figure 23: The ECU Mobile Revealed Bugs

Such a small but heterogeneous variety of models gave the user the ability to detect several problems with GUI (generally minor ones, but highly noticeable). Figure 23 summarizes the all revealed defects, while the entire list of screenshots representing a test report can be found at Appendix B.

First, defects with the sizes of the graphics were noticed. The pirate logo looks correct on the high-dimension screen of the Galaxy Tab, but is cut on smaller resolutions. At the same time, there are layout problems with the logo on the main screen (it is not fully visible and merges in color with button labels in the case of small screens). All these point to inaccurate work with the graphics and screen design during development (e.g., the developer has to provide several variants of the same image with different sizes for different families of screen resolution).

Second, the Wildfire S set up for Russian language helped to reveal bugs with localization. Titles of some buttons were translated, but others were not. Such half-support of different languages points to the low quality of the project organization (localization should be provided fully for selected languages or just disabled for English).

5.1.2 Web Mobile App Testing

The launched in the test script Inner Pirate Network (IPN) web application was tested on a proper work of the beautiful popup menu. This popup menu occurs only in case of a small screen resolution (i.e. IPN on the Galaxy Tab looks different). So the test script is dynamic, it performs different actions with the menu based on retrieved actual sizes of the screen (see Appendix A).

The test case included a re-tap of the menu without selecting any options, while taking screenshots to check the correct drawing. Overall, the screenshots were specified in the test script for three checkpoints. Surprisingly, a rather moderate bug was noticed: the popup menu resists showing up after a re-tap (only a top line is visible, see Figure 23). The follow-up manual check (that is always required in case of any automate testing) proved the defect and explained that it occurs only if the user does not select a menu item. The first tap shows the menu, the second hides it, and the third fails.

5.2 Testing LinkedIn for Android

The second case study was conducted under a more popular and widespread Android application like LinkedIn. It was also assumed to be of a greater quality (to the respect of the size of the development company and long history of versions) and to have more complicated GUI (e.g., animations). The version under testing was also downloaded on March 28, 2013. The following devices were used: Samsung Galaxy Tab 2 7.0, HTC Wildfire S, and Huawei AT&T Fusion 2. This case study had the following aims:

- To demonstrate interactions with mobile peripherals like a photo camera
- To demonstrate multidirectional testing (fixed device against nominal results)

The following subsections explain the steps taken. The test script used for verification can be found in Appendix C and the complete screenshot log is provided in Appendix D.

5.2.1 Interactions with Photo Camera

The first testing of the LinkedIn mobile application was conducted for two of the three devices (Galaxy Tab and Fusion on the same Mac node). The devices for testing were chosen via a manual (concrete) selection in the correspondent tab of the CTOMS Master. Overall, the test script for LinkedIn specified 10 screenshots to taken for correspondent checkpoints. The main functionality under testing was the change of a profile photo, but the test scenario started from the initial login screen (see Appendix C).

To change a profile picture, all devices were properly configured, i.e., a developed *CTOMSFakeCam* Android application was installed and set up as a default camera app on each of them. It provided the ability to test the "take photo" functionality while changing a profile picture. The script anticipates that instead of a typical camera screen, a screen is used with a textbox to enter the image URL (under the focus) and a button "snap" to use this image. After clicking to snap, the activity downloads and returns to the app the specified image as if it was taken from real camera.

Obviously such an approach will not help in the case of advanced applications that use camera API internally (without calls to the default camera app component in the system), e.g., advanced QR-code readers. At the same time, it is useful in case of social networks with profiles, posting of pictures, etc.

Figure 24 shows some of the resulting screenshots. This picture was specified in the test script to be used instead of actually taken photo.

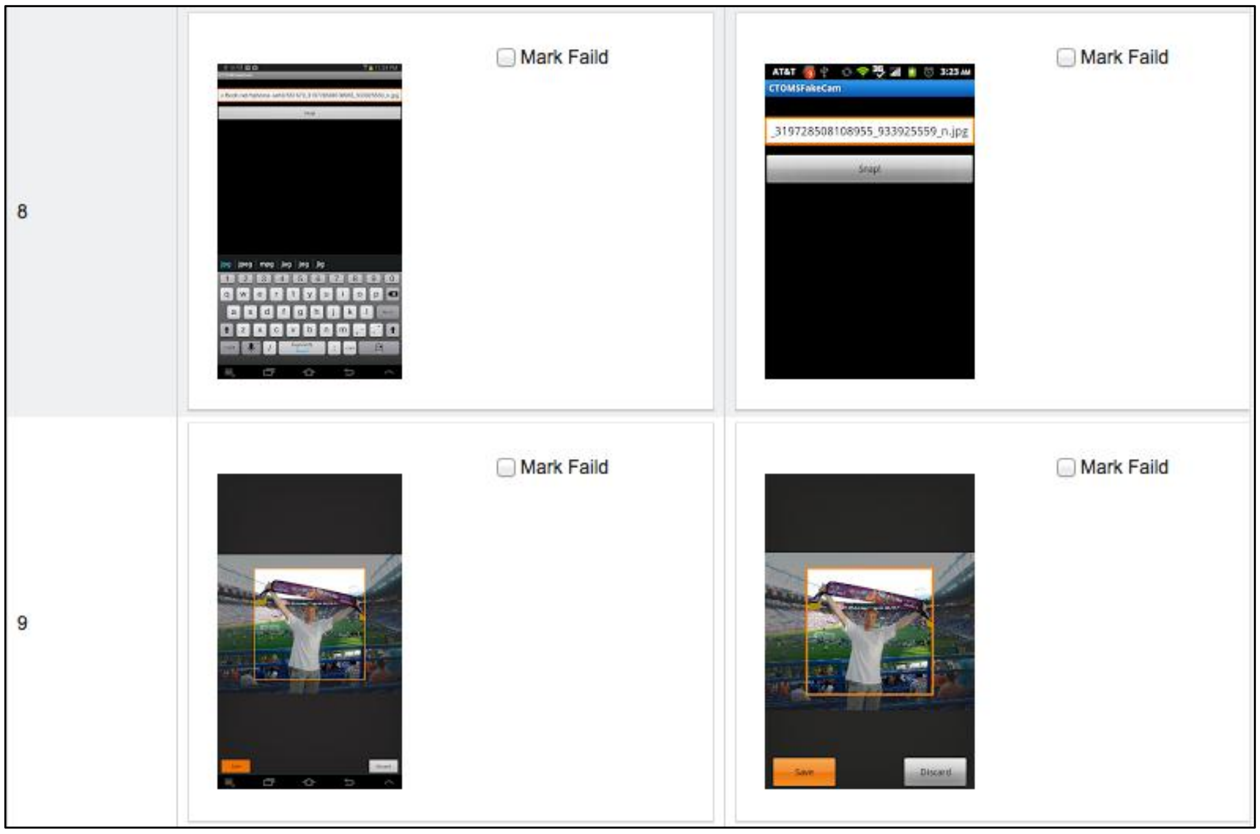
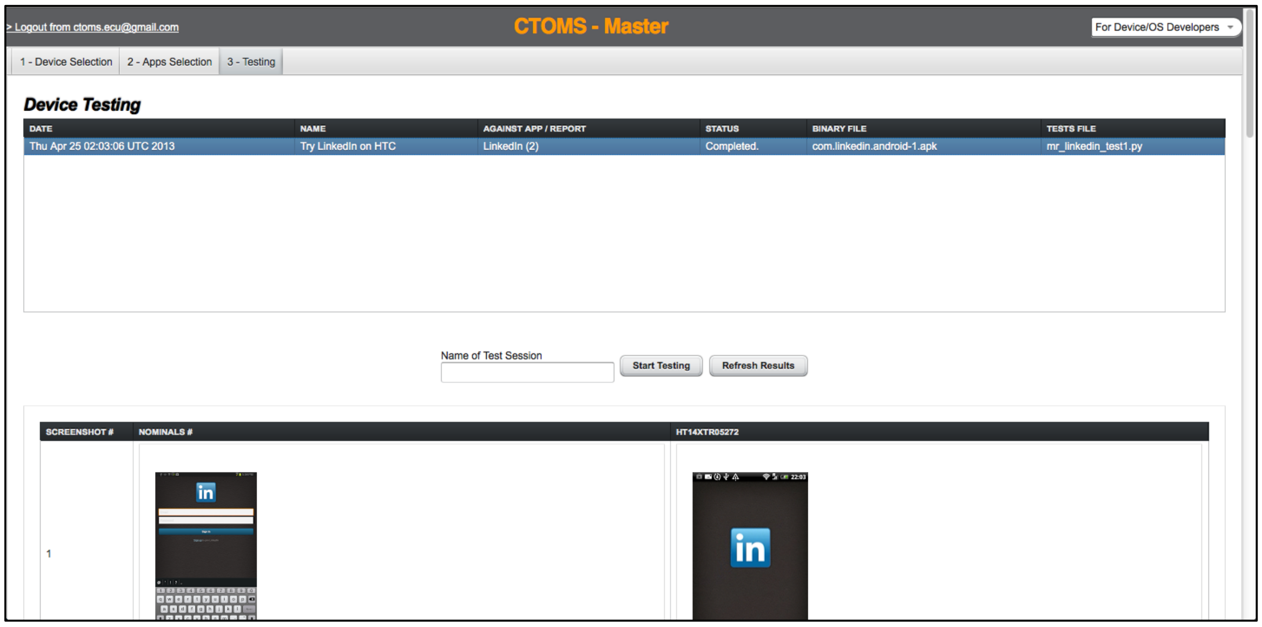


Figure 24: The Interaction with Photo Camera during Testing

Thus the current case study demonstrates the implementation of initial ideas regarding extended control of mobile sensors and peripherals, as well as the general test environment CTOMS smartphones should be placed in.

5.2.2 Multidirectional Testing Demo

Another aspect of this case study was to test multidirectional testing functionality. None of the screenshots taken from Galaxy Tab and Huawei was marked as failed, so the system obtained nominal screenshots. Test sessions of LinkeIn were shared (marked public), and from the "For Device/OS Developers" perspective, the Wildfire S (marked as a hypothetical private new device) was tested against these test sessions. Wildfire S was connected to a separate node under Windows OS. Figure 25 illustrates comparison of newly taken screenshots and previous nominal ones.



Device Testing

DATE	NAME	AGAINST APP / REPORT	STATUS	BINARY FILE	TESTS FILE
Thu Apr 25 02:03:06 UTC 2013	Try LinkedIn on HTC	LinkedIn (2)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py

SCREENSHOT #	NOMINALS #	HT14XTR05272
1		

Figure 25: The Screenshot of a Private (New) Device Testing

5.3 Future Research Using CTOMS

The two main directions of future work with CTOMS are possible: (1) technical improvements and a variety of extensions and (2) conducting research and experiments.

A variety of technical enhancements are possible. For instance, support of other test automation drivers, support of other mobile platforms, support of other testing techniques and test types (integrated statistics, security fuzz testing, static analysis, etc.), support of other mobile sensors, and ability to compare regression screenshots (taken from different builds of the same app) that are currently inactive. Crowdsourcing can be used to intensify CTOMS development as well.

A variety of investigations are possible. Researchers can fix any component of the system (app, device, and OS) or parameter of a smartphone model and investigate mobile testing (e.g., effectiveness of testing) targeted at functional and UI defects. At the same time, different techniques or forms of testing can be applied.

This thesis promotes as the next step more fundamental experiments of the correlation between OS updates and bugs that appeared in the popular and important widely used apps. The inspiration for this direction is provided by experience when a lot of tailoring is needed to an app under development after a new OS version is released, especially in the case of complicated applications with calls to low-level APIs or generally that do not use common approaches.

In the conclusion to this chapter, we can state that the conducted case studies show the applicability of targeted concepts, desired functionality, and chosen design solutions. Additionally, the ability to test interactions with OS was also tested (i.e., uploading of a test script without any application). The CTOMS approach is compatible with most Android devices, but some occasional problems can occur in case of highly tailored and restricted models. For *instance, even with Huawei AT&T Fusion 2, we encountered a problem of a periodic resistance to taking a screenshot using standard Monkeyrunner. The problem is natural, as full control over a device requires root access that is entirely available on development devices but not on production ones. No critical problems were encountered with the efficiency, but they are anticipated with the growth of CTOMS and future enhancements were described previously.*

Finally, directions for future work were listed covering both technical enhancements and empirical research.

CHAPTER 6: CONCLUSION

The work described in this thesis is aimed at improving cloud-based mobile testing and research cloud crowdsourcing in mobile testing. The author hopes that the results provided and contributions will be usefully applied and effectively leveraged for real purposes.

The achievements include (1) an in-depth analysis of the state-of-the-art cloud-based mobile testing, (2) the creation of a methodology of test techniques application for mobile testing over a cloud, (3) the development of the CTOMS framework, and (4) the performance of two demonstrative case studies.

The developed CTOMS platform provides a cloud service to run tests on a variety of remote Android devices using standard Monkeyrunner test scripts that focus on taken screenshots and a comparison of checkpoint screenshots. The platform consists of two parts: the master deployed at Google App Engine cloud and a slave node to be deployed at a participant's site (on a server with connected smartphones). CTOMS provides the implementation for two demanded novel functionalities: multidirectional testing and orientation test techniques' integration. CTOMS provides a user with suggestions about coverage of configurations to test on using combinatorial approaches: base choice, pair-wise, and t-way (using NIST ACTS tool for calculations). The current CTOMS version supports automate functional testing of Android applications and detection of defects in user interfaces (that is highly demanded by modern mobile development).

CTOMS provides ample room for technical extensions and conducted research experiments. Topical directions were analyzed and listed.

Results of this thesis were already reflected in the two papers accepted at the top international conferences in software engineering [164, 165].

REFERENCES

[1] "Apple App Store," https://itunes.apple.com/us/genre/ios/id36?mt=8, accessed April 2013.

[2] "Google Play," https://play.google.com/store/apps, accessed April 2013.

[3] Bank of America. "Mobile Banking," https://www.bankofamerica.com/online-banking/mobile.go, accessed April 2013.

[4] White, J., Clarke, S., Doughtery, B., Thompson, C. & Shmidt, D. (2010, November) "R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services," Springer Journal of Internet Services and Applications, Volume 1, Number 1, pp. 45 - 56.

[5] Carr, D.F. "Hurricane Sandy: Mobile, Social Tools Help Emergency Management," Brainyard, http://www.informationweek.com/thebrainyard/news/social_media_monitoring/240012463/hurricane-sandy-mobile-social-tools-help-emergency-management, accessed April 2013.

[6] Work, D.B. & Bayen, A.M. (2008, November) "Impacts of the Mobile Internet on Transportation Cyberphysical Systems: Traffic Monitoring using Smartphones," in Proceedings of National Workshop for Research on High-Confidence Transportation Cyber-Physical Systems: Automotive, Aviation and Rail, Washington, DC, USA.

[7] Leijdekkers, P. & Gay, V. (2006, June) "Personal Heart Monitoring and Rehabilitation System using Smart Phones," in Proceedings of International Conference on Mobile Business (ICMB 2006), Copenhagen, Denmark, p. 29.

[8] Moser, K. (2012, August) "Improving Work Processes for Nuclear Plants," in Proceedings of American Nuclear Society Utility Working Conference, Hollywood, Florida, USA.

[9] Wasserman, A. (2010, November) "Software engineering issues for mobile application development," in Proceedings of the Workshop on Future of Software Engineering Research at the 18th International Symposium on Foundations of Software Engineering (ACM SIGSOFT), Santa Fe, USA, pp. 397 - 400.

[10] Warren, C. "Facebook Releases New, Faster iOS App," http://mashable.com/2012/08/23/facebook-ios-5/, accessed April 2013.

[11] uTest, Inc. "The Essential Guide to Mobile App Testing (free eBook)," http://www.utest.com/landing-blog/essential-guide-mobile-app-testing, accessed April 2013.

[12] Holler, R. "Mobile Application Development: A Natural Fit with Agile Methodologies," VersionOne, http://www.versionone.com/pdf/mobiledevelopment.pdf, accessed April 2013.

[13] Denman, J. "ALM expert says Agile methods are the way to conquer mobile ALM," http://www.theserverside.com/tip/ALM-expert-says-Agile-methods-are-the-way-to-conquer-mobile-ALM, accessed May2013.

[14] Howe, J. (2006, June). "Crowdsourcing: A Definition," Crowdsourcing Blog.

[15] Rowinski D. “Mobile Carriers and OEMs Get Android App Testing Cloud from Apkudo,” ReadWrite, http://readwrite.com/2012/02/07/mobile-carriers-and-oems-get-a, accessed April 2013.

[16] Kharchenko, V., Siora, O. & Sklyar, V. (2009, February) “Design and testing technique of FPGA-based critical systems,” in Proceedings of CADSM 2009, 10th International Conference - The Experience of Designing and Application of CAD Systems in Microelectronics, Polyana-Svalyava, Ukraine, pp. 305 - 314.

[17] Kuhn, R., Kacker, R., Lei, Y. & Hunter, J. (2009, August) “Combinatorial Software Testing,” IEEE Computer, Volume 42, Number 8, pp. 94 - 96.

[18] Mahmood, R., Esfahani, N., Kacem, T., Mirzaei, N., Malek, S. & Stavrou A. (2012) “A whitebox approach for automated security testing of Android applications on the cloud,” in Proceedings of the 7th International Workshop on Automation of Software Test, pp. 22 - 28.

[19] Windows Phone Dev Center. “Testing Apps for Windows Phone,” http://msdn.microsoft.com/library/windowsphone/develop/jj247547(v=vs.105).aspx, accessed April 2013.

[20] Hu, C. & Neamtiu, I. (2011) “Automating GUI testing for Android applications,” in Proceedings of the 6th International Workshop on Automation of Software Test, ACM New York, NY, USA, pp. 77 - 83.

[21] Kuhn, D.R., Kacker, R.N. & Lei, Y. (2010, October) “Practical Combinatorial Testing,” NIST Special Publication, pp. 13 - 15.

[22] Windows Phone Dev Center. “Test your app’s performance,” http://msdn.microsoft.com/library/windowsphone/develop/jj247547(v=vs.105).aspx#BKMK_testperf, accessed April 2013.

[23] “TestFlight,” https://testflightapp.com, accessed April 2013.

[24] Vilkomir, S. (2012) “Cloud Testing: A State-of-the-Art Review,” Information & Security: An International Journal, Volume 28, Issue 2, Number 17, pp. 213 - 222.

[25] Tilley. S. & Parveen, T. (2012, November) “Software Testing in the Cloud: Perspectives on an Emerging Discipline,” IGI Global.

[26] Tsai, W., Chen, X., Liu, L., Zhao, Y., Tang, L. & Zhao, W. (2010) “Testing as a service over cloud,” in Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering, pp. 181 – 188.

[27] Kalliosaari, L., Taipale, O. & Smolander, K. (2012) “Testing in the Cloud: Exploring the Practice,” IEEE Software, Volume 29, Issue 2, pp. 46 - 51.

[28] Weidong, F. & Yong, X. (2011, August) “Cloud testing: The next generation test technology,” in Proceedings of the 10th International Conference Electronic Measurement & Instruments (ICEMI), Chengdu, China, pp. 291 - 295.

[29] Inçki, K., Ari, I. & Soze, H. (2012) “A Survey of Software Testing in the Cloud,” in Proceedings of the IEEE Sixth International Conference on Software Security and Reliability Companion, pp. 18 - 23.

[30] Priyanka, Chana, I. & Rana, A. (2012, May) “Empirical evaluation of cloud-based testing techniques: a systematic review,” ACM SIGSOFT Software Engineering Notes archive, Volume 37, Issue 3, pp. 1 - 9.

[31] Mote, D. (2011, March) “Cloud based Testing Mobile Apps,” 2nd IndicThreads.com Conference On Software Quality, Pune, India.

[32] Database of Cyber Security and Information Systems Information Analysis Center. “Cloud Testing,” https://sw.thecsiac.com/databases/url/key/7848/8764/8765#.USGPb-h8vDm, accessed April 2013.

[33] Tilley, S. & Parveen, T. (2012, September) “Software Testing in the Cloud: Migration & Execution,” Springer Briefs in Computer Science.

[34] Riungu, L.M., Taipale, O. & Smolander, K. (2010) “Research Issues for Software Testing in the Cloud,” in Proceedings of the IEEE 2nd International Conference Cloud Computing Technology and Science (CloudCom), pp. 557 - 564.

[35] Rhoton, J. & Haukioja, R. (2011, May) “Cloud Computing Architected: Solution Design Handbook,” Recursive, Limited.

[36] Coulouris, G., Dollimore, J., Kindberg, T. & Blair, G. (2011, May) “Distributed Systems: Concepts and Design,” Addison-Wesley, (5th ed.).

[37] Jacobson, D., Brail, G. & Woods, D. (2011, December) “APIs: A Strategy Guide,” O’Reilly Media, Inc., pp. 60 - 70.

[38] Randles, M., Lamb, D. & Taleb-Bendiab, A. (2010) “Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing,” Advanced Information Networking and Applications Workshops (WAINA), IEEE 24th International Conference.

[39] Android Developers. “User Human Interface Guidelines for iOS,” http://developer.android.com/design/get-started/principles.html, accessed April 2013.

[40] Android Developers. “Design Principles for better User Experience,” http://developer.android.com/design/get-started/principles.html, accessed April 2013.

[41] Pretotyping. “Androgen (for Android app generator),” http://www.pretotyping.org/androgen, accessed April 2013.

[42] Muccini, H., Francesco, A. & Esposito, P. (2012) "Software testing of mobile applications: Challenges and future research directions," in Proceedings of the 7th International Workshop on Automation of Software Test (AST).

[43] Franke, D. & Weise, C. (2011, March) "Providing a Software Quality Framework for Testing of Mobile Applications," in Proceedings of the IEEE 4th International Conference on Software Testing, Verification and Validation (ICST), Berlin, Germany, pp. 431 - 434.

[44] Milano D. (2011, June) "Android Application Testing Guide," Publishing Ltd.

[45] Frederick G. & Lal, R. (2009) "Testing a Mobile Web Site," Beginning Smartphone App Development, Part IV, Apress, pp. 259 – 272.

[46] Sama, M., Elbaum, S., Raimondi, F., Rosenblam, D. & Wang, Z. (2010, October) "Context-Aware Adaptive Applications: Fault Patterns and Their Automated Identification," IEEE Transactions on Software Engineering, pp. 644 - 661.

[47] Dantas, V., Marinho, F., Da Costa, A. & Andrade, R. (2009) "Testing requirements for mobile applications," in Proceedings of the 24th International Symposium on Computer and Information Sciences (ISCIS).

[48] Macadamian. "Test Strategies for Smartphones and Mobile Devices," http://www.macadamian.com/images/uploads/whitepapers/MobileTestStrategies_Aug2010.pdf, accessed April 2013.

[49] Perfecto Mobile. "Make Your Mobile Testing Solution Enterprise-Ready," http://www.perfectomobile.com/portal/cms/resources/enterprise-ready_white-paper, accessed April 2013.

[50] Keynote. "Testing Strategies and Tactics for Mobile Applications," http://www.keynote.com/docs/whitepapers/WP_Testing_Strategies.pdf, accessed April 2013.

[51] "Android Developers," http://developer.android.com, accessed April 2013.

[52] Franke D., Elsemann, C. & Kowalewski, S. (2012) "Reverse Engineering and Testing Service Life Cycles of Mobile Platforms," in Proceedings of the 23rd International Workshop on Database and Expert Systems Applications (DEXA), pp. 16 - 20.

[53] Suri, I. "Canonical Unveils the Ubuntu Phone at CES 2013," http://siliconangle.com/blog/2013/01/10/canonical-unveils-the-ubuntu-phone-at-ces-2013/, accessed April 2013.

[54] Haselton, T. "Android Has 56.1% Of Global OS Market Share," Gartner Says, TechnoBuffalo, http://www.technobuffalo.com/2012/05/16/android-has-56-1-of-global-os-market-share-gartner-say s, accessed April 2013.

[55] Fidelman, M. “The Latest Infographics: Mobile Business Statistics For 2012,” Forbes, http://www.forbes.com/sites/markfidelman/2012/05/02/the-latest-infographics-mobile-business-statistics-for-2012, accessed April 2013.

[56] Maji, K., Hao, K., Sultana, S. & Bagchi, S. (2010) “Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian,” in Proceedings of the 21st International Symposium in Software Reliability Engineering (ISSRE), pp. 249 - 258.

[57] AppBrain. “Android Statistics,” http://www.appbrain.com/stats/, accessed April 2013.

[58] “Perfecto Mobile,” http://www.perfectomobile.com, accessed April 2013.

[59] Perfecto Mobile. “UFT Mobile, the official HP mobile testing solution,” http://www.perfectomobile.com/portal/cms/services/uft_mobile, accessed April 2013.

[60] Sterling, C. “Testing devices made easier with Perfecto Mobile's MobileCloud platform and Team Foundation Server,” MSDN Blogs, http://blogs.msdn.com/b/visualstudioalm/archive/2013/02/21/testing-devices-made-easier-with-perfecto-mobile-s-mobilecloud-platform-and-team-foundation-server.aspx, accessed April 2013.

[61] Whiting, R. “New HP Tools Help Developers Test Cloud, Mobile Applications,” http://www.crn.com/news/applications-os/240049875/new-hp-tools-help-developers-test-cloud-mobile-applications.htm, accessed April 2013.

[62] “Keynote DeviceAnywhere,” http://www.keynotedeviceanywhere.com, accessed April 2013.

[63] Galde, J. “Keynote's DeviceAnywhere platform Fully Integrated with RQM 3.0.1,” IBM developerWorks, https://www.ibm.com/developerworks/mydeveloperworks/blogs/deviceanywhere/entry/keynote_s_deviceanywhere_platform_fully_integrated_with_rqm_3_0_12?lang=en, accessed April 2013.

[64] SOASTA. “CloudTest On-Demand,” http://www.soasta.com/services/cloudtest-on-demand, accessed April 2013.

[65] “Cigniti Mobile Testing,” http://www.cigniti.com/mobile-testing, accessed April 2013.

[66] “SeeTest Mobile Cloud,” http://experitest.com/cloud/, accessed April 2013.

[67] Gorilla Logic. “CloudMonkey,” https://www.gorillalogic.com/cloudmonkey, accessed April 2013.

[68] Sauce Labs. “Appium on Sauce,” https://saucelabs.com/appium, accessed April 2013.

[69] SeleniumHQ. “Selenium RC,” http://docs.seleniumhq.org/projects/remote-control/, accessed April 2013.

[70] “Jenkins Continuous Integration,” http://jenkins-ci.org, accessed April 2013.

[71] BitBar. “Testdroid Cloud,” http://testdroid.com/product/testdroid-cloud, accessed April 2013.

[72] “Scirocco Cloud,” http://www.scirocco-cloud.com/en/, accessed April 2013.

[73] “LessPainful,” https://www.lesspainful.com, accessed April 2013.

[74] “Calabash-Android,” https://github.com/calabash/calabash-android, accessed April 2013.

[75] Bsquare. “TestQuest 10,” http://www.bsquare.com/products/testquest-automated-testing-platform/testquest-10, accessed April 2013.

[76] Zap-Fix. “ZPX for Mobile,” http://www.zaptechnologies.com/products/zap-fix/portfolio/zap-fix-for-mobile, accessed April 2013.

[77] “Jamo Solutions Technology,” http://www.jamosolutions.com/technology.html, accessed April 2013.

[78] “Apkudo,” http://www.apkudo.com, accessed April 2013.

[79] “Datum,” http://www.metricowireless.com/systems/measdata.cfm, accessed April 2013.

[80] Android Developers. “UI/Application Exerciser Monkey tool,” http://developer.android.com/tools/help/monkey.html, accessed April 2013.

[81] Konstantinidis, A., Costa, C., Larkou, G. & Zeinalipour-Yazti, D. (2012) “Demo: a programming cloud of smartphones,” in Proceedings of the 10th international conference on Mobile systems, applications, and services, pp. 465 - 466.

[82] Turner, H., White, J., Reed, J., Galindo, J., Porter, A., Marathe, M, Vullikanti, A. & Gokhale, A. (2012, November) “Building a Cloud-Based Mobile Application Testbed,” IGI Global, pp. 382 - 403.

[83] She, S., Sivapalan, S. & Warren, I. (2009, April) “Hermes: A Tool for Testing Mobile Device Applications,” in Proceedings of the Software Engineering Conference (ASWEC), Queensland, Australia.

[84] Robotium. “User Scenario Testing for Android,” http://code.google.com/p/robotium/, accessed April 2013.

[85] JetBrains. “YouTrack,” http://www.jetbrains.com/youtrack, accessed April 2013.

[86] “IBM RQM,” http://www-142.ibm.com/software/products/us/en/ratiqualmana, accessed April 2013.

[87] GurockSoftware. “TestRails,” http://www.gurock.com/testrail/, accessed April 2013.

[88] “Assembla,” https://www.assembla.com, accessed April 2013.

[89] “TestFlight,” https://testflightapp.com, accessed April 2013.

[90] “Air On App,” http://www.aironapp.com, accessed April 2013.

[91] “Launchpad,” http://launchpadapp.com, accessed April 2013.

[92] “HokeyApp,” http://hockeyapp.net, accessed April 2013.

[93] Flower M. “Continuous Integration,” http://martinfowler.com/articles/continuousIntegration.html, accessed April 2013.

[94] “Google Analytics,” http://www.google.com/analytics, accessed April 2013.

[95] Perfecto Mobile. “MobileCloud Monitoring,” http://www.perfectomobile.com/portal/cms/services/monitoring.html, accessed April 2013.

[96] “User Testing,” http://www.usertesting.com/mobile, accessed April 2013.

[97] Amazon Mobile Build Distribution. “Amazon A/B testing for Android,” https://developer.amazon.com/sdk/ab-testing/reference/how-ab-works.html, accessed April 2013.

[98] “FluidUI,” https://www.fluidui.com, accessed April 2013.

[99] “Kickfolio,” http://kickfolio.com, accessed April 2013.

[100] “Pidoco,” https://pidoco.com/en/tour/why-pidoco, accessed April 2013.

[101] Android Developers. “Monkeyrunner,” http://developer.android.com/tools/help/monkeyrunner_concepts.html, accessed April 2013.

[102] TestPlant. “EggPlant for Mobile,” http://www.testplant.com/products/eggplant/mobile/, accessed April 2013.

[103] Whittaker, J., Arbon, J. & Carollo, J. (2012, April) “How Google Tests Software,” Addison-Wesley Professional.

[104] Nextag Mobile. “Nextag Shopping,” https://play.google.com/store/apps/details?id=com.nextag.android&hl=en, accessed April 2013.

[105] “SandStrom,” http://sandstorm.impetus.com/, accessed April 2013.

[106] “NeoLad,” http://www.neotys.com/product/mobile-load-testing.html, accessed April 2013.

[107] “Checkmarks,” http://www.checkmarx.com, accessed April 2013.

[108] “Veracode,” http://www.veracode.com/products/mobile-application-security.html, accessed April 2013.

[109] uTest. "Mobile Security Testing," http://www.utest.com/mobile-security-testing, accessed April 2013.

[110] Android Developers. "Security Tips," http://developer.android.com/training/articles/security-tips.html, accessed April 2013.

[111] "Keynote DeviceAnywhere Test Planner," http://www.keynotedeviceanywhere.com/mobile-test-planner.html, accessed April 2013.

[112] "Hexawise," http://hexawise.com, accessed April 2013.

[113] "Android SDK," http://developer.android.com/sdk/, accessed April 2013.

[114] "AndroidViewClient," Extension to Monkeyrunner, https://github.com/dtmilano/AndroidViewClient, accessed April 2013.

[115] "ViewServer," https://github.com/romainguy/ViewServer, accessed April 2013.

[116] Android Developers. "UIAutomator," http://developer.android.com/tools/help/uiautomator/, accessed April 2013.

[117] "Robotium," http://code.google.com/p/robotium/, accessed April 2013.

[118] Nagowah, L. & Sowamber, G. (2012, June) "A Novel Approach of Automation Testing on Mobile Devices," in Proceedings of 2012 International Conference on Computer & Information Science (ICCIS), Volume 2, pp. 924 – 930.

[119] Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X. & Bringas, P. (2012) "On the Automatic Categorisation of Android Applications," in Proceedings of the 9th Annual IEEE Consumer Communications ad Networking Conference – Security and Content Protections.

[120] Allevato, A. & Edwards, S. (2012) "RoboLIFT: simple GUI-based unit testing of student-written android applications (abstract only," in Proceedings of the 43rd ACM technical symposium on Computer Science Education, p. 670.

[121] Mirzaei, N., Malek, S., Păsăreanu, C., Esfahani, N. & Mahmood, R. (2012, November) "Testing Android Apps Through Symbolic Execution," ACM SIGSOFT Software Engineering Notes archive, Volume 37, Issue 6, pp. 1 – 5.

[122] Zivkov, D. (2012, May) Touch screen mobile application as part of testing and verification system," in Proceedings of the 35th International Convention MIPRO 2012, pp. 892 - 895.

[123] Amalfitano, D., Fasolino, A., Tramontana, P. & De Carmine, S. (2012, September) "Using GUI ripping for automated testing of Android application," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Germany.

[124] Amalfitano, D., Fasolino, A.R. & Tramontana, P. (2011) "A GUI Crawling-Based Technique for Android Mobile Application Testing," in Proceedings of the Software Testing, Verification and

Validation Workshops (ICSTW), pp. 252 - 261.

[125] Ridene, Y. & Barbier, F. (2011) “A model-driven approach for automating mobile applications testing,” in Proceedings of the 5th European Conference on Software Architecture: Companion, Volume Article No. 9.

[126] Kropp, M. & Morales, P. (2010) “Automated GUI Testing on the Android Platform,” IMVS Fokus Report, Volume 4, Issue 1, p. 33.

[127] Gilbert, P., Chun, B., Cox, L. & Jung, J. “Automating Privacy Testing of Smartphone Applications,” Technical Report CS-2011-02.

[128] Edmondson, J., Gokhale, A. & Sandeep Neema. (2011, December) “Automating Testing of Service-oriented Mobile Applications with Distributed Knowledge and Reasoning,” in Proceedings of the Service-Oriented Computing and Applications (SOCA), pp. 1 - 4.

[129] Takala, T., Katara, M. & Harty, J. (2011) “Experiences of System-Level Model-Based GUI Testing of an Android Application,” in Proceedings of the Software Testing, Verification and Validation (ICST), pp. 377 – 386.

[130] Selvam, R. & Karthikeyani V. (2011, April) “Mobile Software Testing – Automated Test Case Design Strategies,” International Journal on Computer Science and Engineering (IJCSE).

[131] Payet, E. & Spoto, F. (2011) “Static Analysis of Android Programs,” Lecture Notes in Computer Science, Volume 6803, Automated Deduction CADE-23, pp. 439 - 445.

[132] Sadeh, B., Ørbekk, K., Eide, M., Gjerde, N., Tønnesland, T. & Gopalakrishnan, S. (2011) “Towards Unit Testing of User Interface Code for Android Mobile Applications,” Communications in Computer and Information Science, Volume 181, Part 1, pp. 163 – 175.

[133] Liu, Z., Gao, X. & Long, X. (2010, April) “Adaptive random testing of mobile application,” in Proceedings of Computer Engineering and Technology (ICCET), Volume 2, pp. 297 – 301.

[134] Abdallah, N. & Ramakrishnan, S. (2009) “Automated Stress Testing of Windows Mobile GUI Applications,” in Proceedings of the 20th International Symposium on Software Reliability Engineering.

[135] Sivapalan, S. & Warren, I. (2009) “Hermes: A Tool for Testing Mobile Device Applications,” in Proceedings of Software Engineering Conference ASWEC '09, Australia.

[136] Zhi-fang Liu, Bin Liu & Xiao-peng Gao. (2009) “SOA based mobile application software test framework,” in Proceedings of the 8th International Conference on Reliability, Maintainability and Safety ICRMS 2009, pp. 765 - 769.

[137] Long Xiang, Gao Xiaopeng & Jiang Bo (2007, May) “MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices,” in Proceedings of 2nd International Workshop on Automation of Software Test AST '07.

[138] Russino, A. & Santoro, C. (2007) “Remote Evaluation of Mobile Applications,”. Lecture Notes in Computer Science, Volume 4849/2007, pp. 155-169.

[139] Frey, H., Görgen, D., Lehnert, J. & Sturm, P. (2004) “A Java-Based Uniform Workbench for Simulating and Executing Distributed Mobile Applications,” Lecture Notes in Computer Science, Volume 2952, Scientific Engineering of Distributed Java Applications, pp. 116 - 127.

[140] NIST. “Combinatorial Methods in Software Testing,” http://csrc.nist.gov/groups/SNS/acts/, accessed April 2013.

[141] Grindal, M., Offutt, J. & Andler, S.F. (2005, March) “Combination Testing Strategies: a Survey,” Software Testing, Verification and Reliability, Volume 15, Number 3, pp. 167 - 199.

[142] Lei, Y. & Tai, K.C. (1998, november) “In-parameter-order: a Test Generation Strategy for Pairwise Testing,” in Proceedings of 3rd IEEE Int. High-Assurance Systems Engineering Symp., IEEE Press, pp. 254 - 261.

[143] Kuhn, D.R., Lei, Y. & Kacker, R. (2008, June) “Practical Combinatorial Testing - Beyond Pairwise,” IEEE IT Professional, pp. 19 - 23.

[144] Maximoff, J.R., Trela, M.D., Kuhn, D.R. & Kacker, R. (2010, April) “A Method for Analyzing System State-space Coverage within a t-Wise Testing Framework,” IEEE International Systems Conference 2010, San Diego.

[145] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V. & Lawrence, J. (2007, March) “IPOG: A General Strategy for T-Way Software Testing,” in Proceeding of ECBS '07, IEEE Engineering of Computer Based Systems conference, pp. 549 - 556.

[146] Chilenski, J. J. & Miller, S. (1994, September) “Applicability of Modified Condition/Decision Coverage to Software Testing,” Software Engineering Journal, pp. 193 - 200.

[147] Vilkomir, S. & Bowen, J. P. (2006) “From MC/DC to RC/DC: Formalization and Analysis of Control-flow Testing Criteria, Formal Aspects of Computing,” Volume 18, Number 1, pp. 42 - 62.

[148] NIST. “ACTS tool,” http://csrc.nist.gov/groups/SNS/acts/download, accessed April 2013.

[149] Bach, J. “ALLPAIRS Test Case Generation Tool (Version 1.2.1),” http://www.satisfice.com/tools/pairs.zip, accessed April 2013.

[150] Google App Engine. “Developers portal,” https://developers.google.com/appengine, accessed April 2013.

[151] “Localtunnel,” http://progrium.com/localtunnel/, accessed April 2013.

[152] “Forwardhq,” https://forwardhq.com, accessed April 2013.

[153] Android Developers. “Platform Versions,” http://developer.android.com/about/dashboards/, accessed April 2013.

[154] Android Developers. "Screen Sizes and Densities," http://developer.android.com/about/dashboards/, accessed April 2013.

[155] VAADIN. "Google App Engine Integration", https://vaadin.com/book/-/page/advanced.gae.html, accessed April 2013.

[156] "Dropbox," https://www.dropbox.com, accessed April 2013.

[157] AndroidViewClient. "Secure Mode," https://github.com/dtmilano/AndroidViewClient/wiki/Secure-mode, accessed April 2013.

[158] "FakeCamera," https://github.com/bryanl/FakeCamera, accessed April 2013.

[159] "VAADIN framework," https://vaadin.com/, accessed April 2013.

[160] "Derby database", http://db.apache.org/derby/, accessed April 2013.

[161] "Jersey JAX_RS," http://jersey.java.net/, accessed April 2013.

[162] "Hazelcast," http://www.hazelcast.com/, accessed April 2013.

[163] "Protocol Buffers," https://developers.google.com/protocol-buffers/, accessed April 2013.

[164] Starov, O. & Vilkomir, S. (2013, May) "Integrated TaaS Platform for Mobile Development: Architecture Solutions," in Proceedings of the Eighth International Workshop on Automation of Software Test (AST 2013), San Francisco, USA, in conjunction with the 35th International Conference on Software Engineering (ICSE 2013).

[165] Starov, O., Vilkomir, S. & Kharchenko, V. (2013, July) "Cloud Testing for Mobile Software Systems: Concept and Prototyping," in Proceedings of the Eighth International Conference on Software Engineering and Applications (ICSOFT-EA 2013), Reykjavík, Iceland, as a part of the 8th International Joint Conference on Software Technologies (ICSOFT 2013).

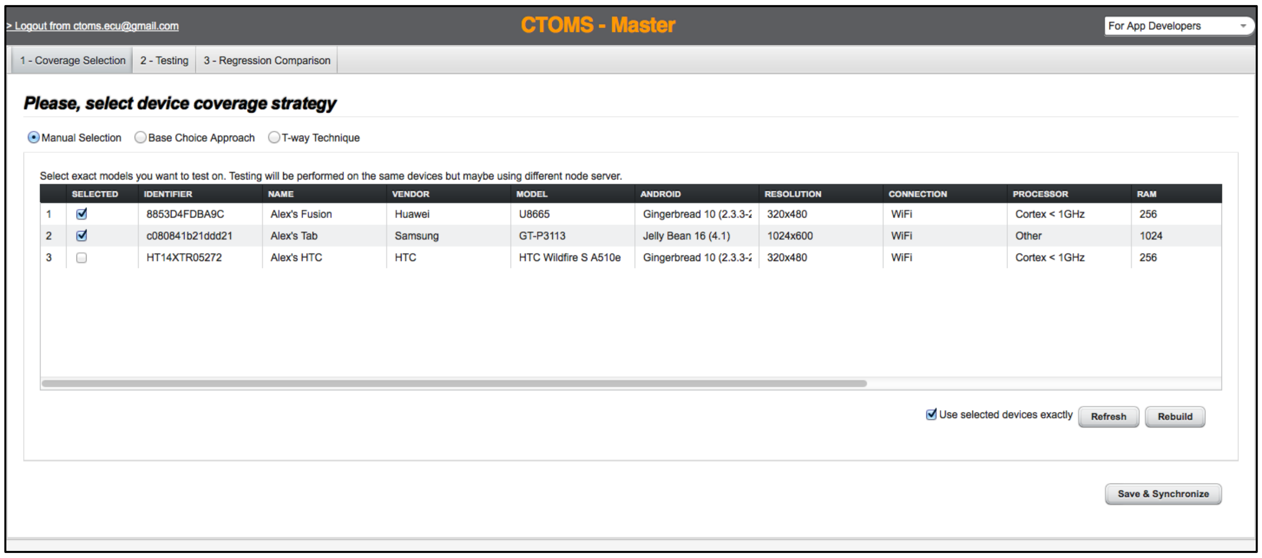
APPENDIX A: TEST SCRIPT FOR ECU MOBILE

```
import sys
# Imports the monkeyrunner modules used by this program
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice

# REQUIRED: Function to get and save screenshots is used
def takeScreenshot(device, file):
	MonkeyRunner.sleep(1.0)
	result = device.takeSnapshot()
	MonkeyRunner.sleep(1.0)
	result.writeToFile(file, 'png')
	MonkeyRunner.sleep(1.0)

# REQUIRED: Test script connects only to the one current device
device = MonkeyRunner.waitForConnection()

# Installs the Android package
device.installPackage('com.blackboard.android.central.ecu-1.apk')

# Installation can take a time
MonkeyRunner.sleep(5)

# Runs the start activity
package = 'com.blackboard.android.central.ecu'
activity = 'com.blackboard.android.central.activity.SpringboardActivity'
device.startActivity(component = package + '/' + activity)

# Launching can take a time
MonkeyRunner.sleep(2)

# CHECKPOINT: Splash screen
takeScreenshot(device, 'ecu_test1_s0.png')

MonkeyRunner.sleep(2)

# Get device properties
# REQUIRED: height and width are predefined names of variables
# that will be duplicated for each device
height = int(device.getProperty('display.height'))
width = int(device.getProperty('display.width'))

# CHECKPOINT: Main screen 1
takeScreenshot(device, 'ecu_test1_s1.png')

# Scrolling right
device.drag((width - width / 10, 3 * height / 4), (width / 10, 3 * height / 4),
1, 1)
MonkeyRunner.sleep(2)
```

```
# CHECKPOINT: Main screen 2
takeScreenshot(device, 'ecu_test1_s2.png')

# Because after scrolling
device.touch(60, 120, MonkeyDevice.DOWN_AND_UP)

# Browser needs time
MonkeyRunner.sleep(10)

# CHECKPOINT: Inner Pirate Network
takeScreenshot(device, 'ecu_test1_s3.png')

# Only small screens will have the menu popup!
device.touch(width - 20, height / 20, MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(5)

# CHECKPOINT: Menu - 1
takeScreenshot(device, 'ecu_test1_s4.png')

# Should hide on second tap
device.touch(width - 20, height / 20, MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(5)

# CHECKPOINT: Menu - 2
takeScreenshot(device, 'ecu_test1_s5.png')

# Should appear again
device.touch(width - 20, height / 20, MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(5)

# CHECKPOINT: Menu - 3
takeScreenshot(device, 'ecu_test1_s6.png')

# REQUIRED: Presses the Home button
device.press('KEYCODE_HOME', MonkeyDevice.DOWN_AND_UP)

MonkeyRunner.sleep(2)

# REQUIRED: Removes the app
device.removePackage('com.blackboard.android.central.ecu')
```

APPENDIX B: SCREENSHOTS OF ECU MOBILE TESTING

1) Shows the whole list of shared devices in the system.



Please, select device coverage strategy

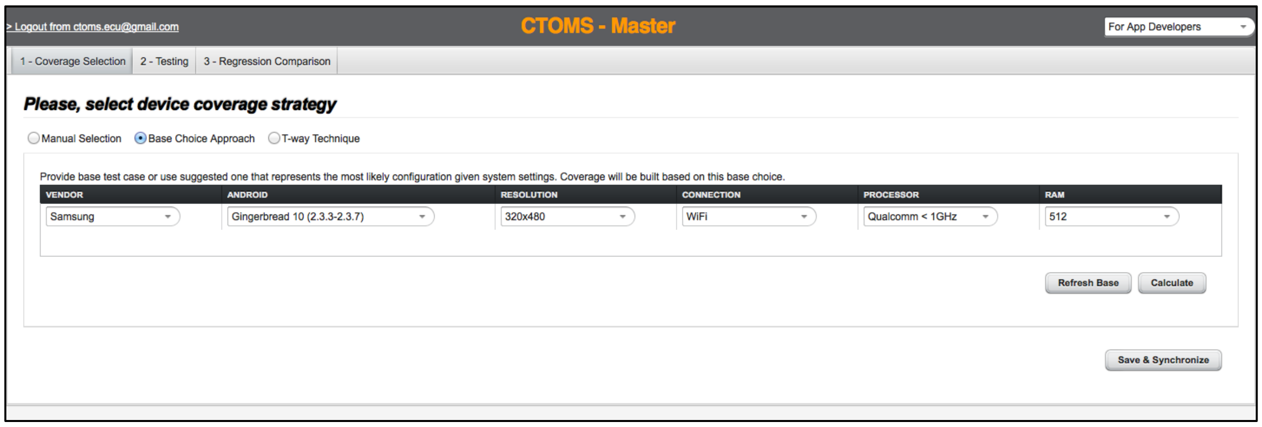
[✓ Manual Selection] Base Choice Approach T-way Technique

Select exact models you want to test on. Testing will be performed on the same devices but maybe using different node server.

	SELECTED	IDENTIFIER	NAME	VENDOR	MODEL	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM
1	✓	8853D4FDBA9C	Alex's Fusion	Huawei	U8665	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256
2	✓	c080841b21ddd21	Alex's Tab	Samsung	GT-P3113	Jelly Bean 16 (4.1)	1024x600	WiFi	Other	1024
3		HT14XTR05272	Alex's HTC	HTC	HTC Wildfire S A510e	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256

✓ Use selected devices exactly

2) Shows possibility to use *base choice* combinatorial approach.



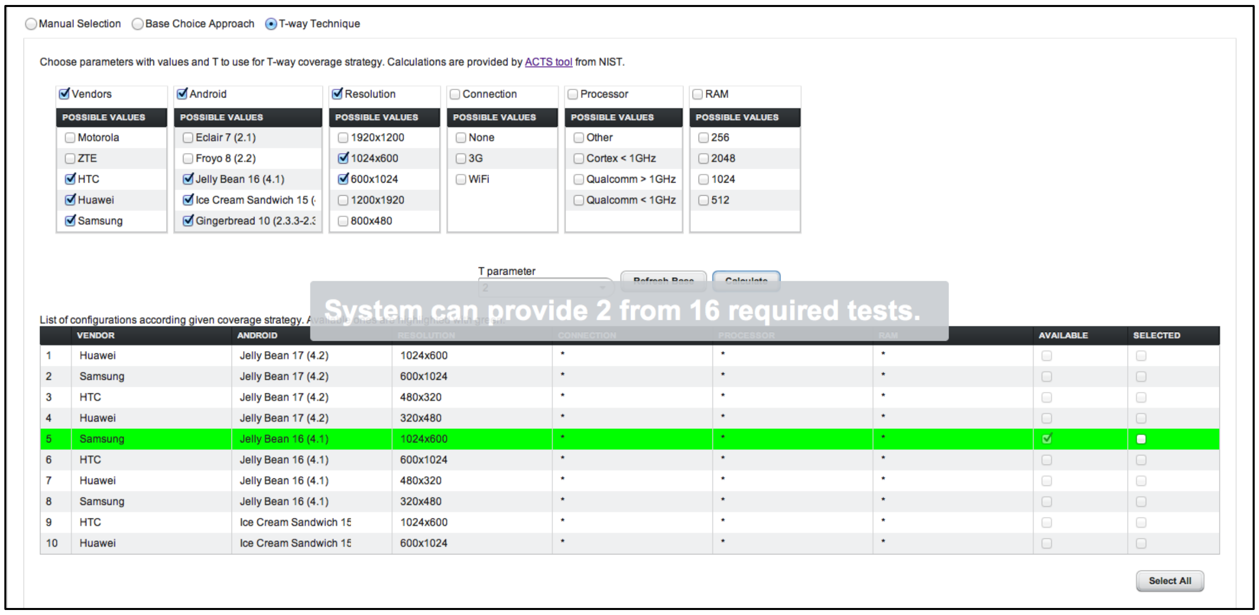
Please, select device coverage strategy

Manual Selection [✓ Base Choice Approach] T-way Technique

Provide base test case or use suggested one that represents the most likely configuration given system settings. Coverage will be built based on this base choice.

VENDOR	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM
Samsung	Gingerbread 10 (2.3.3-2.3.7)	320x480	WiFi	Qualcomm < 1GHz	512

3) Shows usage of *pair-wise* coverage calculation.


Manual Selection | Base Choice Approach | [✓ T-way Technique]

Choose parameters with values and T to use for T-way coverage strategy. Calculations are provided by ACTS tool from NIST.

✓ Vendors	✓ Android	✓ Resolution	Connection	Processor	RAM
POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES
Motorola	Eclair 7 (2.1)	1920x1200	None	Other	256
ZTE	Froyo 8 (2.2)	✓ 1024x600	3G	Cortex < 1GHz	2048
✓ HTC	✓ Jelly Bean 16 (4.1)	✓ 600x1024	WiFi	Qualcomm > 1GHz	1024
✓ Huawei	✓ Ice Cream Sandwich 15 (	1200x1920		Qualcomm < 1GHz	512
✓ Samsung	✓ Gingerbread 10 (2.3.3-2.3	800x480			

T parameter 2 | Refresh Base | Calculate

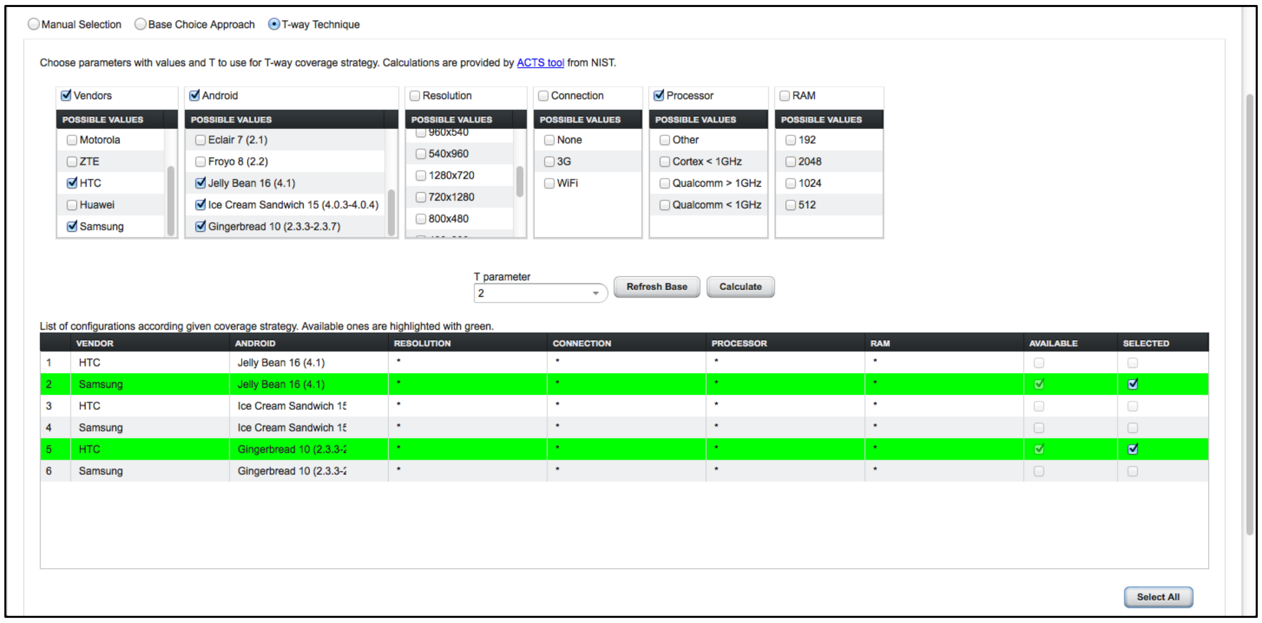
System can provide 2 from 16 required tests.

List of configurations according given coverage strategy. Available ones are highlighted with green.

	VENDOR	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM	AVAILABLE	SELECTED
1	Huawei	Jelly Bean 17 (4.2)	1024x600	*	*	*		
2	Samsung	Jelly Bean 17 (4.2)	600x1024	*	*	*		
3	HTC	Jelly Bean 17 (4.2)	480x320	*	*	*		
4	Huawei	Jelly Bean 17 (4.2)	320x480	*	*	*		
5	Samsung	Jelly Bean 16 (4.1)	1024x600	*	*	*	✓	
6	HTC	Jelly Bean 16 (4.1)	600x1024	*	*	*		
7	Huawei	Jelly Bean 16 (4.1)	480x320	*	*	*		
8	Samsung	Jelly Bean 16 (4.1)	320x480	*	*	*		
9	HTC	Ice Cream Sandwich 15	1024x600	*	*	*		
10	Huawei	Ice Cream Sandwich 15	600x1024	*	*	*		

Select All

4) Shows selection of all available devices according to provided coverage.


Manual Selection | Base Choice Approach | [✓ T-way Technique]

Choose parameters with values and T to use for T-way coverage strategy. Calculations are provided by ACTS tool from NIST.

✓ Vendors	✓ Android	Resolution	Connection	✓ Processor	RAM
POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES	POSSIBLE VALUES
		960x540			
Motorola	Eclair 7 (2.1)	540x960	None	Other	192
ZTE	Froyo 8 (2.2)	1280x720	3G	Cortex < 1GHz	2048
✓ HTC	✓ Jelly Bean 16 (4.1)	720x1280	WiFi	Qualcomm > 1GHz	1024
Huawei	✓ Ice Cream Sandwich 15 (4.0.3-4.0.4)	800x480		Qualcomm < 1GHz	512
✓ Samsung	✓ Gingerbread 10 (2.3.3-2.3.7)				

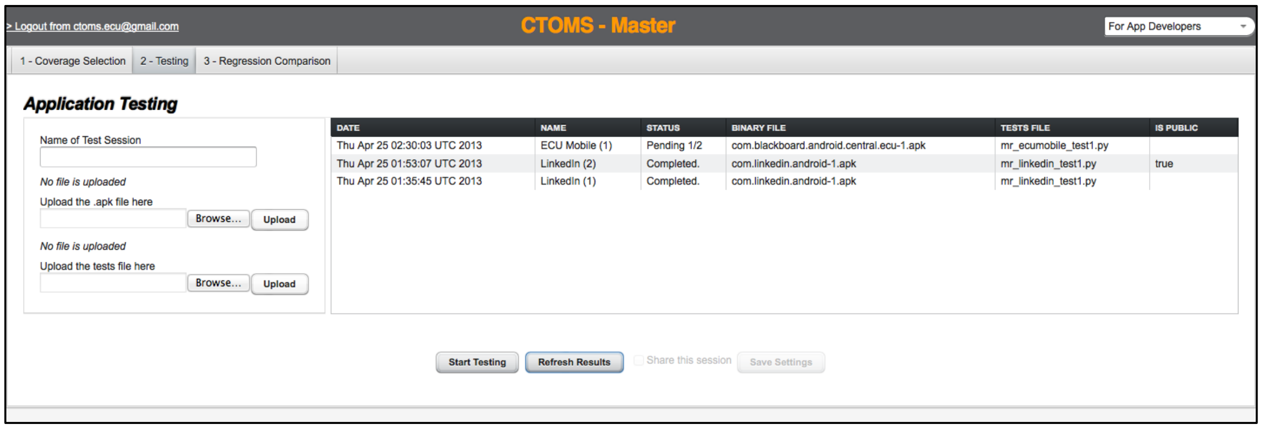
T parameter 2 | Refresh Base | Calculate

List of configurations according given coverage strategy. Available ones are highlighted with green.

	VENDOR	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM	AVAILABLE	SELECTED
1	HTC	Jelly Bean 16 (4.1)	*	*	*	*		
2	Samsung	Jelly Bean 16 (4.1)	*	*	*	*	✓	✓
3	HTC	Ice Cream Sandwich 15	*	*	*	*		
4	Samsung	Ice Cream Sandwich 15	*	*	*	*		
5	HTC	Gingerbread 10 (2.3.3-2	*	*	*	*	✓	✓
6	Samsung	Gingerbread 10 (2.3.3-2	*	*	*	*		

Select All

5) Shows test session in progress (results from two devices are expected).


> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For App Developers

1 - Coverage Selection | 2 - Testing | 3 - Regression Comparison

Application Testing

Name of Test Session

No file is uploaded

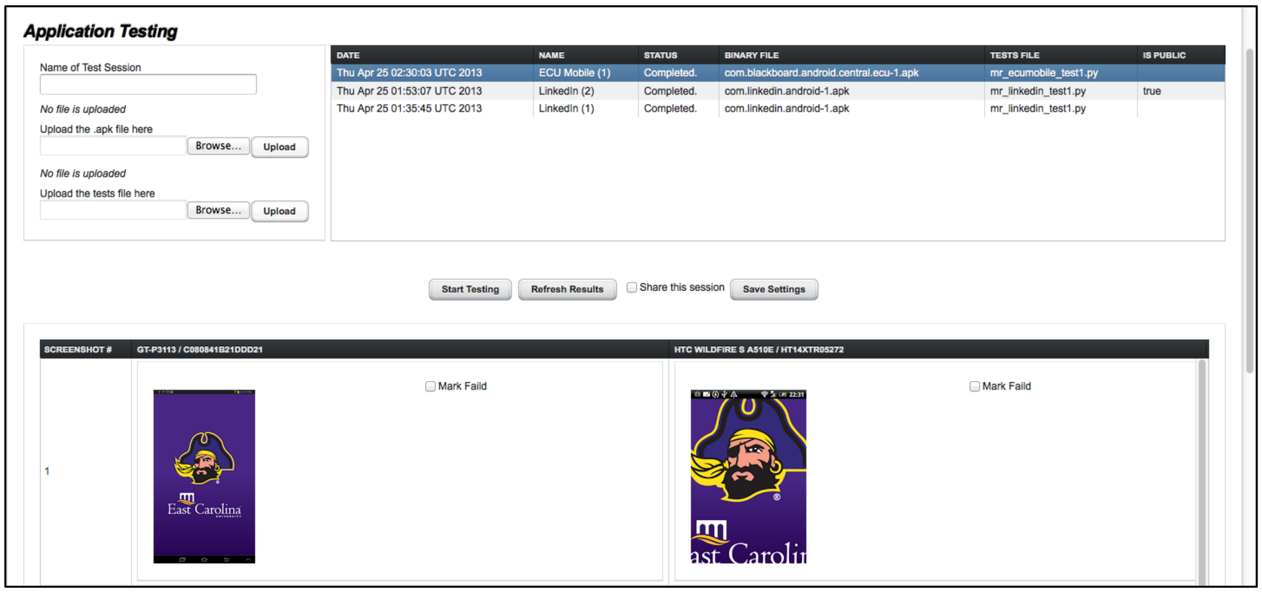
Upload the .apk file here

No file is uploaded

Upload the tests file here

DATE	NAME	STATUS	BINARY FILE	TESTS FILE	IS PUBLIC
Thu Apr 25 02:30:03 UTC 2013	ECU Mobile (1)	Pending 1/2	com.blackboard.android.central.ecu-1.apk	mr_ecumobile_test1.py	
Thu Apr 25 01:53:07 UTC 2013	LinkedIn (2)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	true
Thu Apr 25 01:35:45 UTC 2013	LinkedIn (1)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	

6) Shows finished test session (and a problem with a splash screen).


Application Testing

Name of Test Session

No file is uploaded

Upload the .apk file here

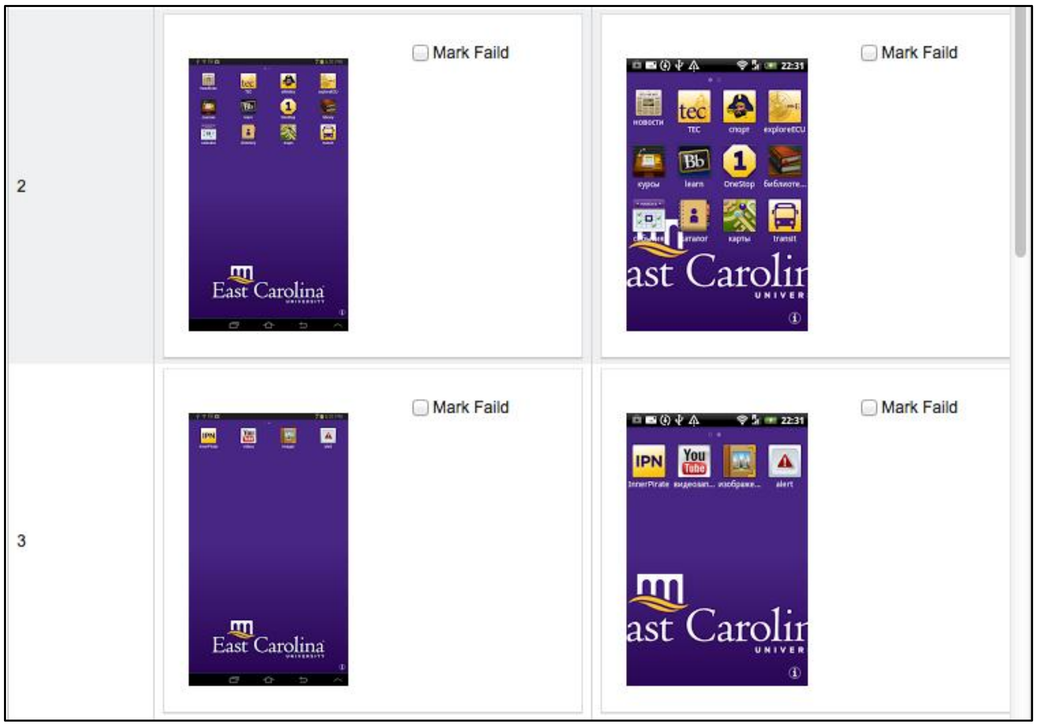
No file is uploaded

Upload the tests file here

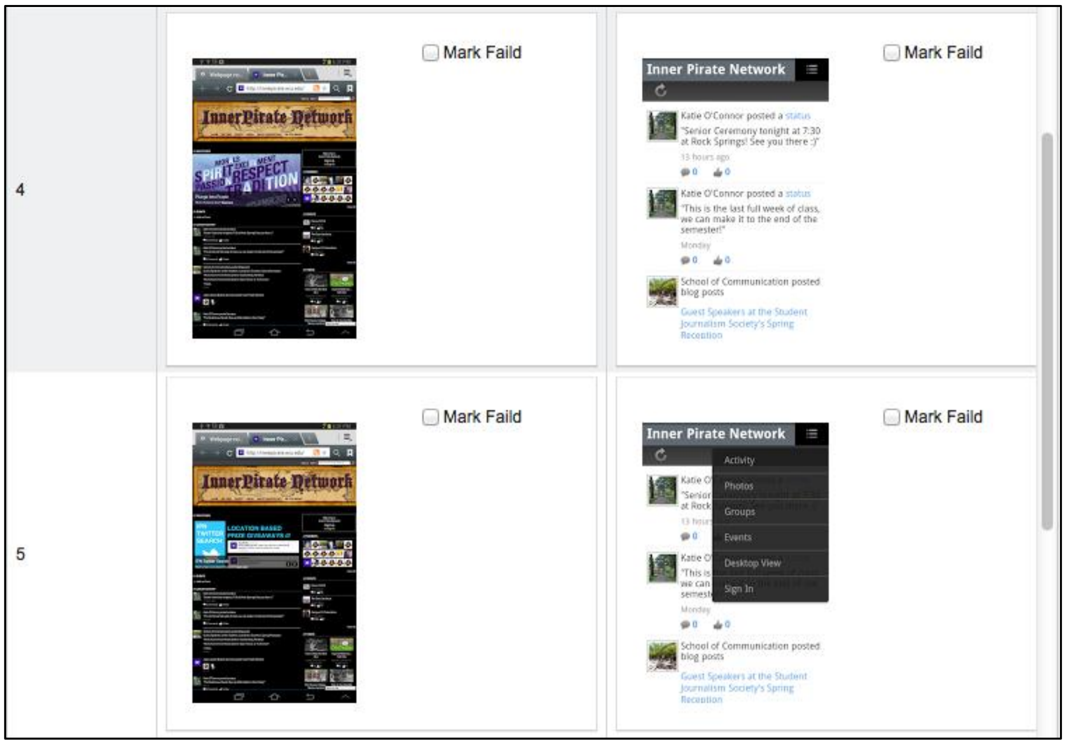
DATE	NAME	STATUS	BINARY FILE	TESTS FILE	IS PUBLIC
Thu Apr 25 02:30:03 UTC 2013	ECU Mobile (1)	Completed.	com.blackboard.android.central.ecu-1.apk	mr_ecumobile_test1.py	
Thu Apr 25 01:53:07 UTC 2013	LinkedIn (2)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	true
Thu Apr 25 01:35:45 UTC 2013	LinkedIn (1)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	

SCREENSHOT #	GT-P3113 / C080841B21DDD21	HTC WILDFIRE S A510E / HT14XTR05272
1	Mark Faild	Mark Faild

7) Shows next two screenshots from the test report (problems with logo and localization).



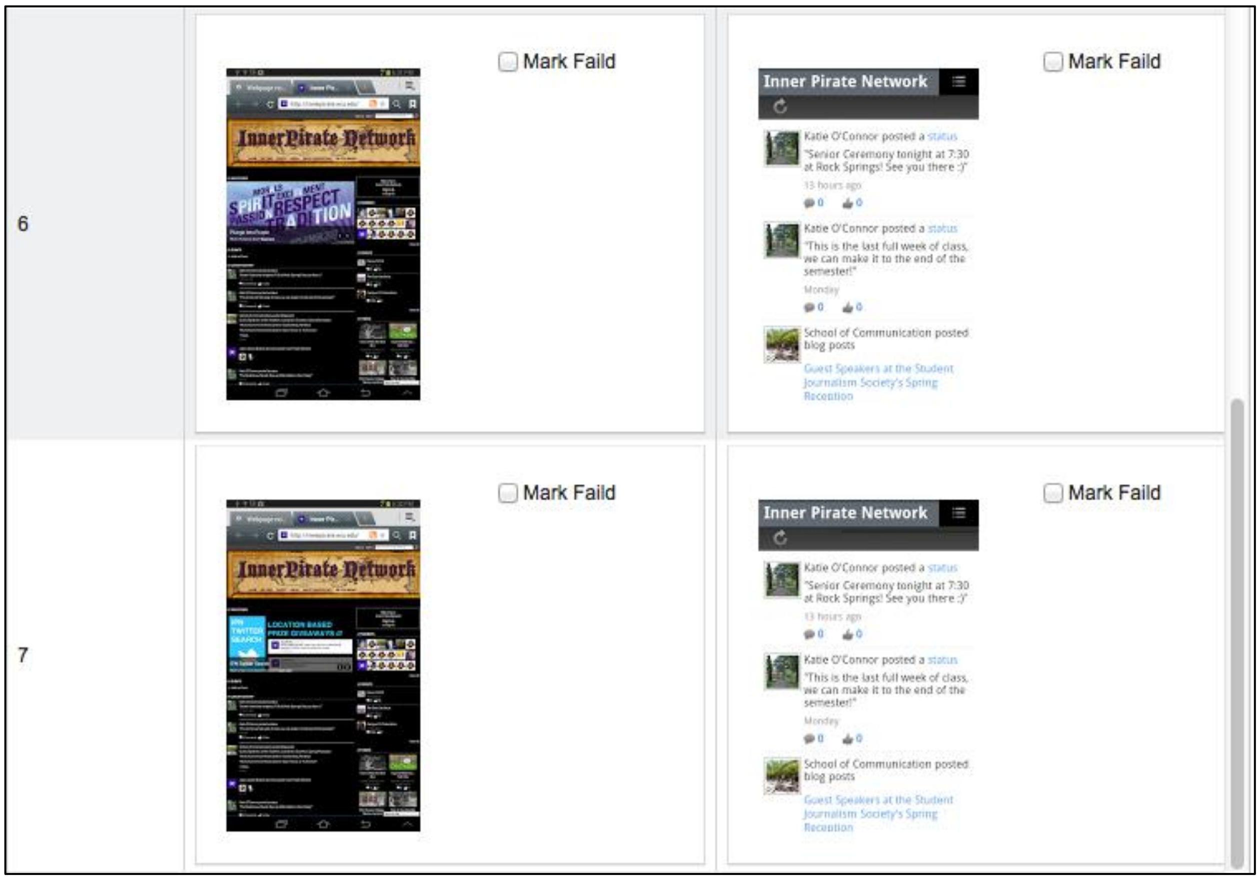
8) Shows next four screenshots from the test report (popup menu is present only in case of small screen resolution, i.e., dynamic web interface).



Katie O'Connor posted a status
"Senior Ceremony tonight at 7:30 at Rock Springs! See you there :)"
13 hours ago

Katie O'Connor posted a status
"This is the last full week of class, we can make it to the end of the semester!"
Monday

School of Communication posted blog posts
Guest Speakers at the Student Journalism Society's Spring Reception



6

Mark Faild

Inner Pirate Network

Katie O'Connor posted a status
"Senior Ceremony tonight at 7:30 at Rock Springs! See you there :)"
13 hours ago
0 0

Katie O'Connor posted a status
"This is the last full week of class, we can make it to the end of the semester!"
Monday
0 0

School of Communication posted blog posts
Guest Speakers at the Student Journalism Society's Spring Reception

Mark Faild

7

Mark Faild

Inner Pirate Network

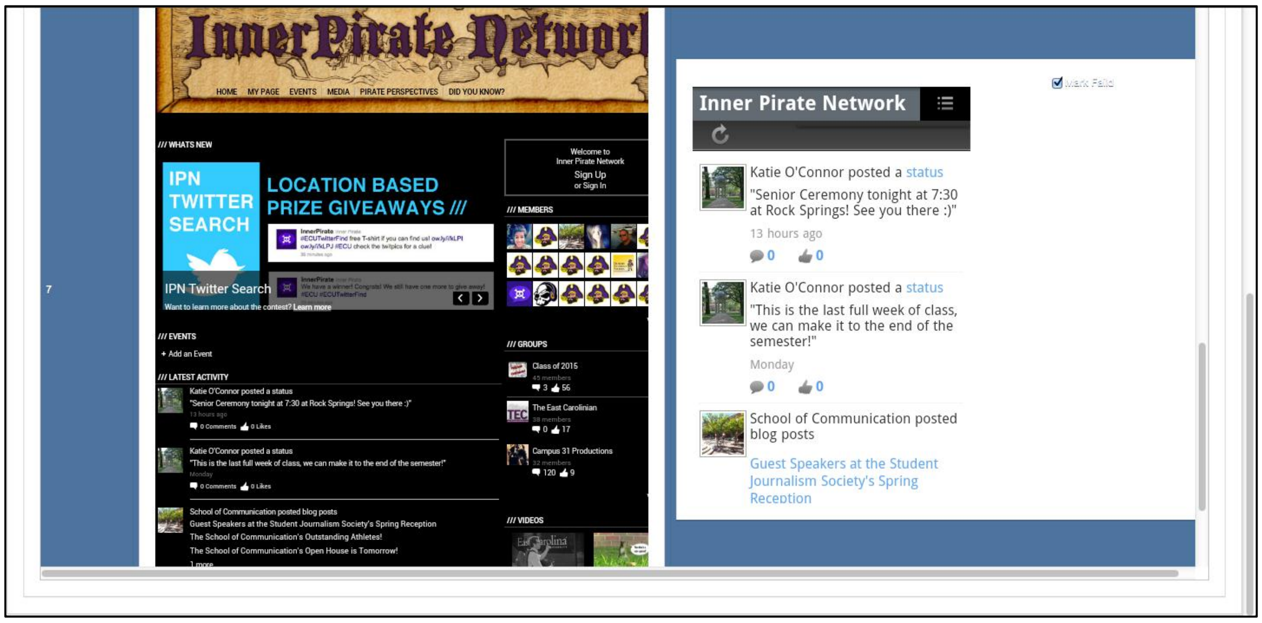
Katie O'Connor posted a status
"Senior Ceremony tonight at 7:30 at Rock Springs! See you there :)"
13 hours ago
0 0

Katie O'Connor posted a status
"This is the last full week of class, we can make it to the end of the semester!"
Monday
0 0

School of Communication posted blog posts
Guest Speakers at the Student Journalism Society's Spring Reception

Mark Faild

9) Shows original-sized screenshots, highlighting the bug with a popup menu (failed).



7

InnerPirate Network

HOME MY PAGE EVENTS MEDIA PIRATE PERSPECTIVES DID YOU KNOW?

/// WHATS NEW

IPN TWITTER SEARCH

LOCATION BASED PRIZE GIVEAWAYS ///

IPN Twitter Search

Want to learn more about the contest? Learn more

Welcome to Inner Pirate Network
Sign Up
or Sign In

/// MEMBERS

/// EVENTS

+ Add an Event

/// LATEST ACTIVITY

Katie O'Connor posted a status
"Senior Ceremony tonight at 7:30 at Rock Springs! See you there :)"
13 hours ago
0 Comments 0 Likes

Katie O'Connor posted a status
"This is the last full week of class, we can make it to the end of the semester!"
Monday
0 Comments 0 Likes

School of Communication posted blog posts
Guest Speakers at the Student Journalism Society's Spring Reception
The School of Communication's Outstanding Athletes!
The School of Communication's Open House is Tomorrow!
1 more

/// GROUPS

Class of 2015
45 members
3 56

The East Carolinian
19 members
0 17

Campus 31 Productions
12 members
120 9

/// VIDEOS

Inner Pirate Network

Katie O'Connor posted a status
"Senior Ceremony tonight at 7:30 at Rock Springs! See you there :)"
13 hours ago
0 0

Katie O'Connor posted a status
"This is the last full week of class, we can make it to the end of the semester!"
Monday
0 0

School of Communication posted blog posts
Guest Speakers at the Student Journalism Society's Spring Reception

☑ Mark Faild

APPENDIX C: TEST SCRIPT FOR LINKEDIN

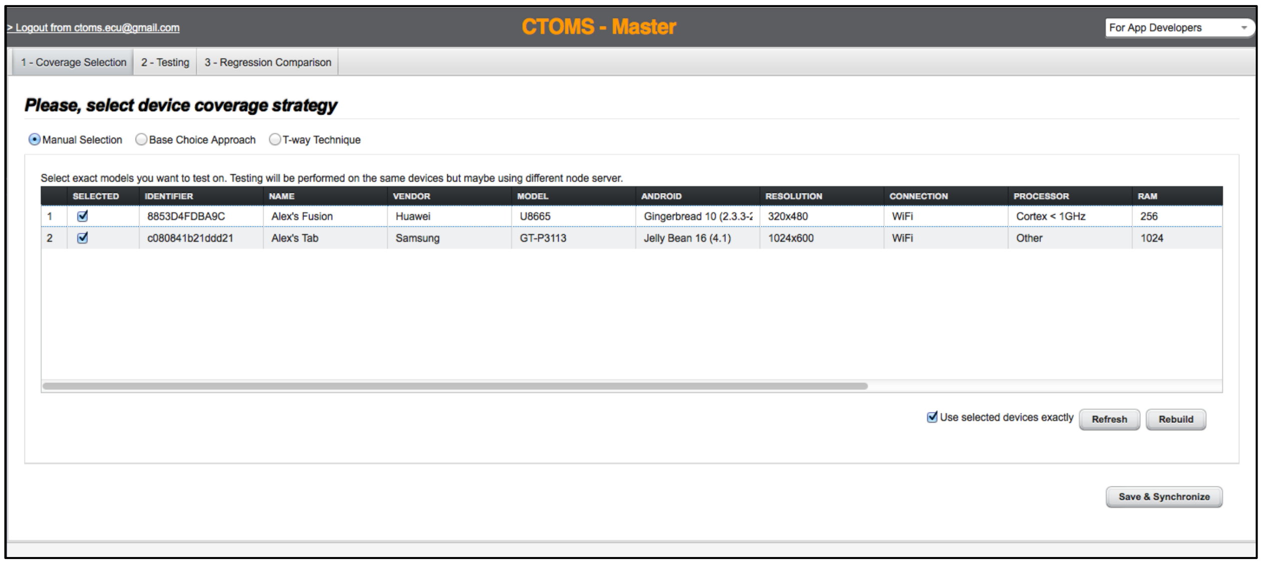
```
import sys
# Imports the monkeyrunner modules used by this program
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice

# REQUIRED: Function to get and save screenshots is used
def takeScreenshot(device, file):
      MonkeyRunner.sleep(1.0)
      result = device.takeSnapshot()
      MonkeyRunner.sleep(1.0)
      result.writeToFile(file, 'png')
      MonkeyRunner.sleep(1.0)

# REQUIRED: Test script connects only to the one current device
device = MonkeyRunner.waitForConnection()

# Installs the Android package
device.installPackage('com.linkedin.android-1.apk')

# Installation can take a time
MonkeyRunner.sleep(5)

# Runs the start activity
package = 'com.linkedin.android'
activity = '.authenticator.LaunchActivity'
device.startActivity(component = package + '/' + activity)

# Launching can take a time
MonkeyRunner.sleep(5)

# Get device properties
# REQUIRED: height and width are predefined names of variables
# that will be duplicated for each device
height = int(device.getProperty('display.height'))
width = int(device.getProperty('display.width'))

# CHECKPOINT: Login screen - 1
takeScreenshot(device, 'linkedin_test1_s1.png')

# Types login
device.type('a_starov@hotmail.com')
device.press('KEYCODE_DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)
# Types password
device.type('qazxsw')
device.press ('KEYCODE_DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

# CHECKPOINT: Login screen - 2
takeScreenshot(device, 'linkedin_test1_s2.png')
```

```
# Presses login button
device.press("DPAD_CENTER", MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(5)

# CHECKPOINT: Sync dialog
takeScreenshot(device, 'linkedin_test1_s3.png')

device.press('KEYCODE_DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
device.press('KEYCODE_DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
device.press("DPAD_CENTER", MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

# CHECKPOINT: Calendar dialog
takeScreenshot(device, 'linkedin_test1_s4.png')

device.press("DPAD_CENTER", MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

# CHECKPOINT: Main screen
takeScreenshot(device, 'linkedin_test1_s5.png')

# Goes to menu
device.touch(20, 50, MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

# CHECKPOINT: Menu (settings)
takeScreenshot(device, 'linkedin_test1_s6.png')

# Goes to edit profile
device.touch(width / 2 + width / 4, height / 2 - height / 4, MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

# CHECKPOINT: Edit profile screen
takeScreenshot(device, 'linkedin_test1_s7.png')

# Goes to edit profile
device.touch(90, 185, MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

device.press('KEYCODE_DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
device.press("DPAD_CENTER", MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(2)

# CAMERA: Types URL for image to use
device.type('https://sphotos-b.xx.fbcdn.net/hphotos-ash3/551679_3197285081089
55_933925559_n.jpg')

# CHECKPOINT: Taken photo
takeScreenshot(device, 'linkedin_test1_s8.png')
```

```
device.press('KEYCODE_DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
device.press("DPAD_CENTER", MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(5)

# CHECKPOINT: Back to profile
takeScreenshot(device, 'linkedin_test1_s9.png')

# Accepts photo
device.press("DPAD_CENTER", MonkeyDevice.DOWN_AND_UP)
MonkeyRunner.sleep(5)

# CHECKPOINT: Back to profile
takeScreenshot(device, 'linkedin_test1_s10.png')

MonkeyRunner.sleep(2)

# REQUIRED: Presses the Home button
device.press('KEYCODE_HOME', MonkeyDevice.DOWN_AND_UP)

MonkeyRunner.sleep(2)

# REQUIRED: Removes the app
device.removePackage('com.linkedin.android')
```

APPENDIX D: SCREENSHOTS OF LINKEDIN TESTING

1) Shows device selection to test LinkedIn on.



> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For App Developers

1 - Coverage Selection | 2 - Testing | 3 - Regression Comparison

Please, select device coverage strategy

[● Manual Selection] ○ Base Choice Approach ○ T-way Technique

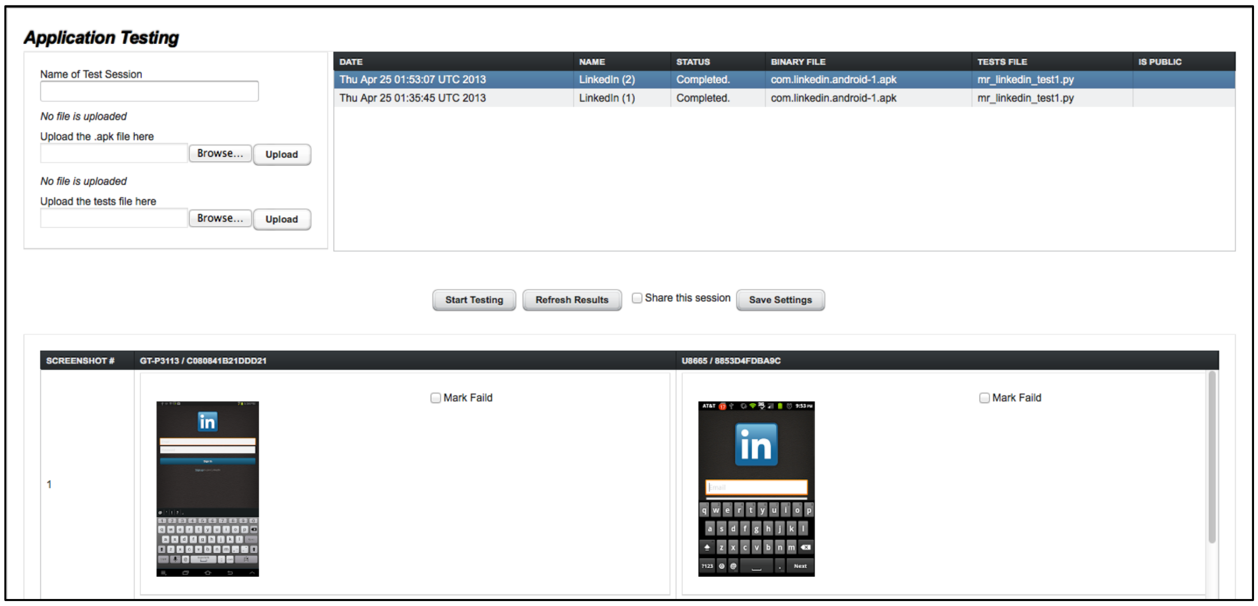
Select exact models you want to test on. Testing will be performed on the same devices but maybe using different node server.

	SELECTED	IDENTIFIER	NAME	VENDOR	MODEL	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM
1	✓	8853D4FDBA9C	Alex's Fusion	Huawei	U8665	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256
2	✓	c080841b21ddd21	Alex's Tab	Samsung	GT-P3113	Jelly Bean 16 (4.1)	1024x600	WiFi	Other	1024

[✓ Use selected devices exactly] Refresh Rebuild

Save & Synchronize

2) Shows completed test session.



Application Testing

Name of Test Session

No file is uploaded

Upload the .apk file here

Browse... Upload

No file is uploaded

Upload the tests file here

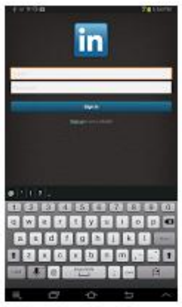
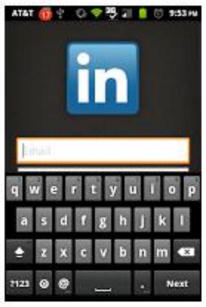
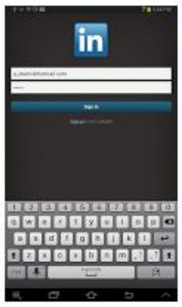
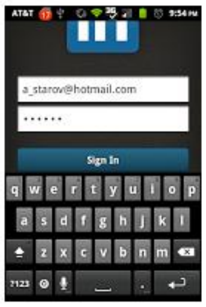
Browse... Upload

DATE	NAME	STATUS	BINARY FILE	TESTS FILE	IS PUBLIC
Thu Apr 25 01:53:07 UTC 2013	LinkedIn (2)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	
Thu Apr 25 01:35:45 UTC 2013	LinkedIn (1)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	

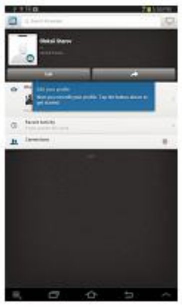
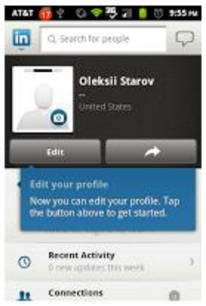
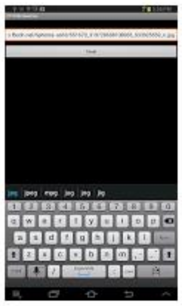
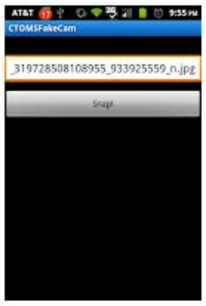
Start Testing | Refresh Results | ☐ Share this session | Save Settings

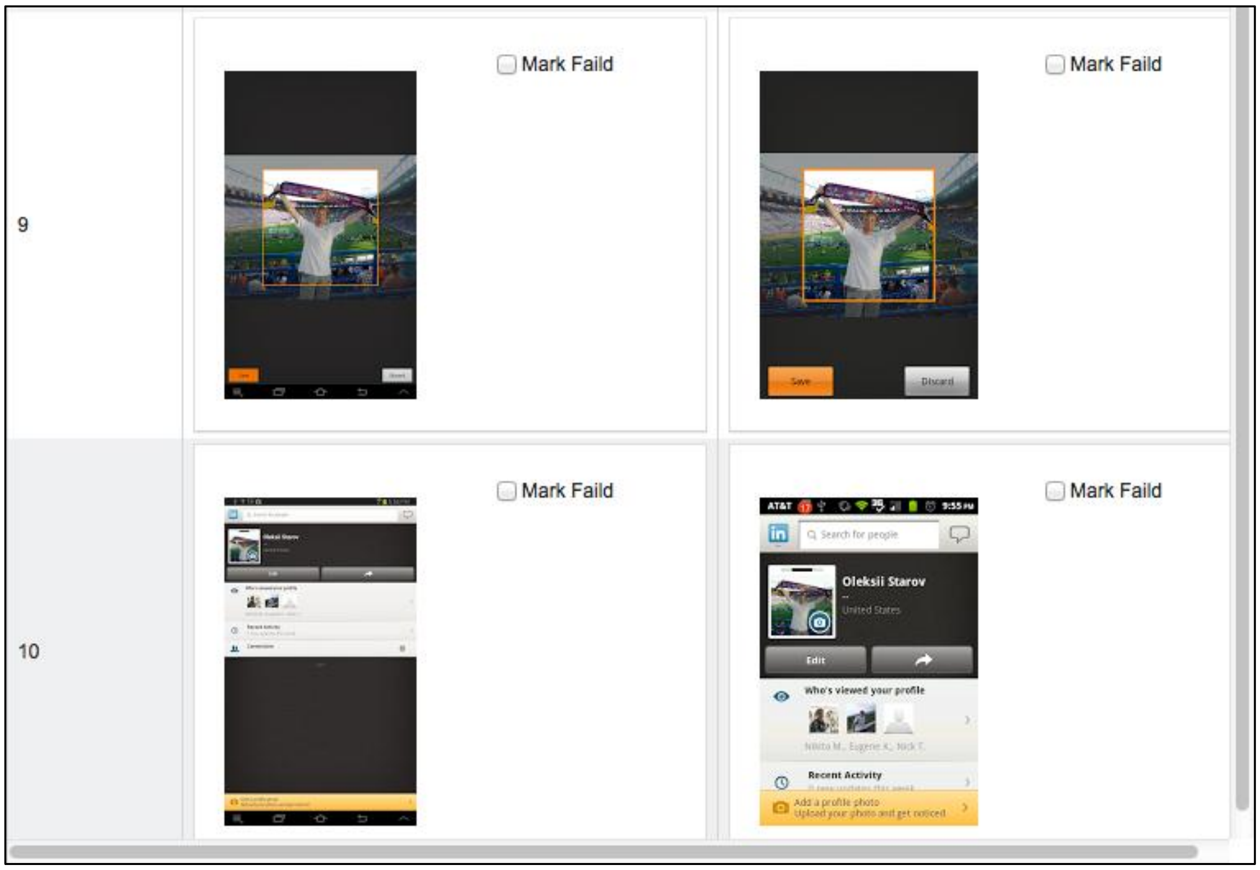
SCREENSHOT #	GT-P3113 / C080841B21DDD21	U8665 / 8853D4FDBA9C
1	☐ Mark Faild	☐ Mark Faild

3) Shows screenshot results for login.

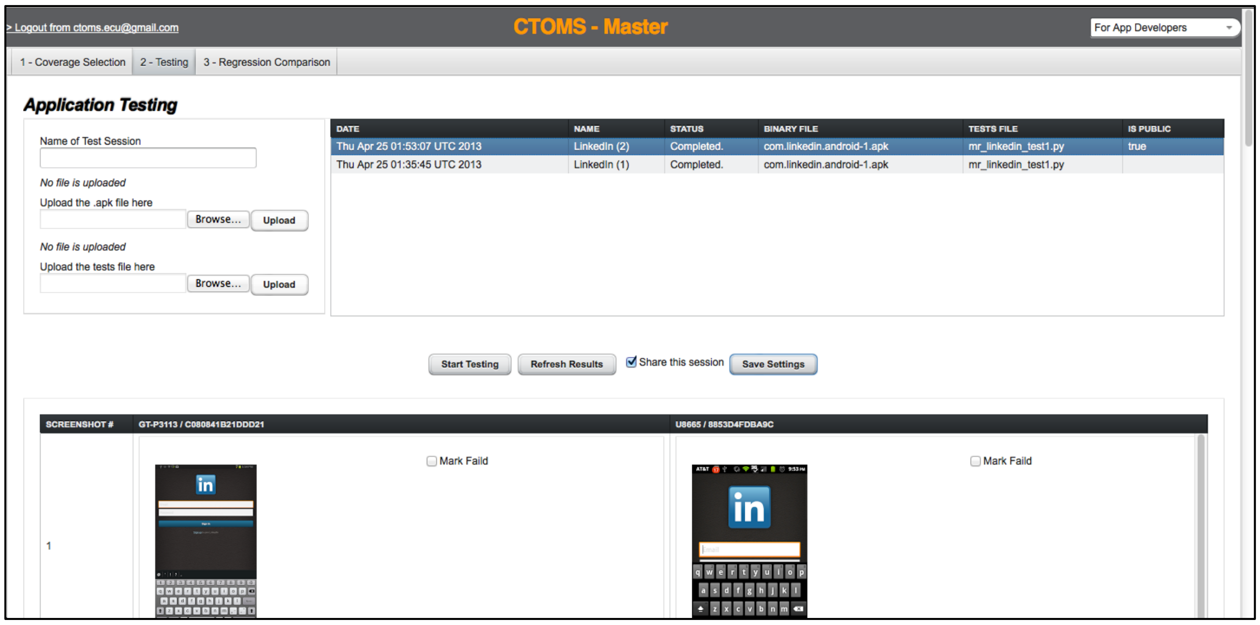
SCREENSHOT #	GT-P3113 / C080841B21DDD21		U8665 / 8853D4FDBA9C	
1		☐ Mark Faild	 Email	☐ Mark Faild
2		☐ Mark Faild	 a_starov@hotmail.com Sign In	☐ Mark Faild

4) Shows screenshot results for profile picture change.

7		☐ Mark Faild	 Oleksii Starov, United States. Edit your profile: Now you can edit your profile. Tap the button above to get started. Recent Activity: 0 new updates this week. Connections	☐ Mark Faild
8		☐ Mark Faild	 CTOMSFakeCam _319728508108955_933925559_n.jpg Snap!	☐ Mark Faild



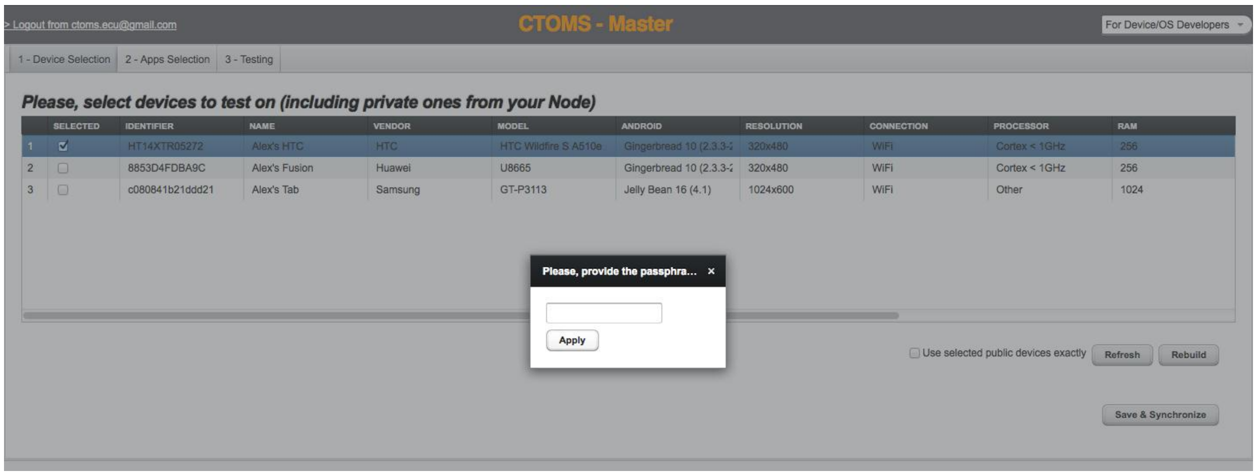
5) Shows how test session was made public.



DATE	NAME	STATUS	BINARY FILE	TESTS FILE	IS PUBLIC
Thu Apr 25 01:53:07 UTC 2013	LinkedIn (2)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	true
Thu Apr 25 01:35:45 UTC 2013	LinkedIn (1)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py	

[✓ Share this session]

6) Shows selection of the private HTC device from device developer's perspective.


> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For Device/OS Developers

1 - Device Selection | 2 - Apps Selection | 3 - Testing

Please, select devices to test on (including private ones from your Node)

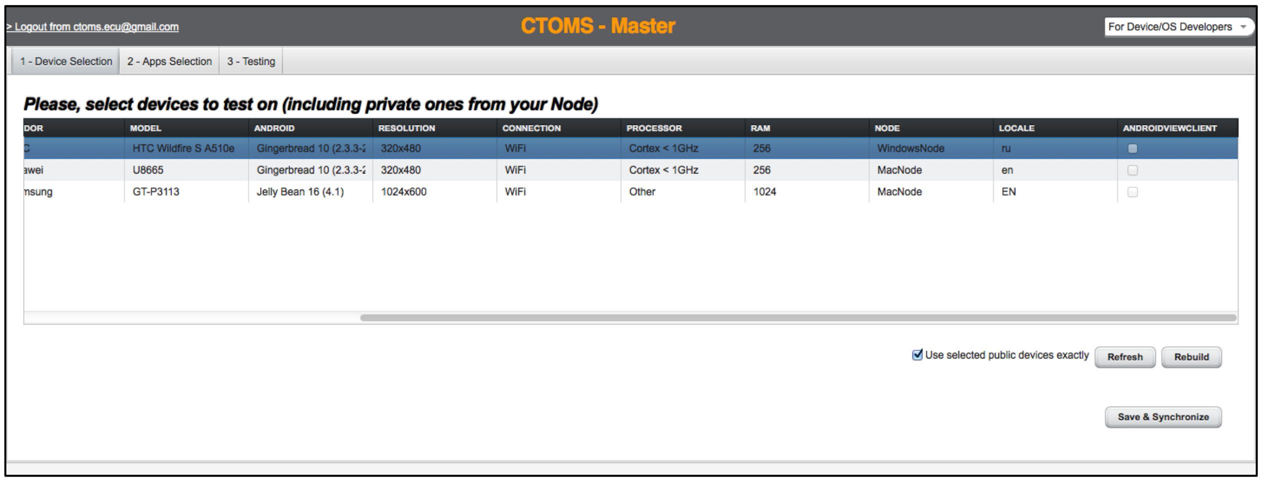
	SELECTED	IDENTIFIER	NAME	VENDOR	MODEL	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM
1	☑	HT14XTR05272	Alex's HTC	HTC	HTC Wildfire S A510e	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256
2	☐	8853D4FDBA9C	Alex's Fusion	Huawei	U8665	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256
3	☐	c080841b21ddd21	Alex's Tab	Samsung	GT-P3113	Jelly Bean 16 (4.1)	1024x600	WiFi	Other	1024

Please, provide the passphra... ×

Apply

☐ Use selected public devices exactly Refresh Rebuild

Save & Synchronize


> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For Device/OS Developers

1 - Device Selection | 2 - Apps Selection | 3 - Testing

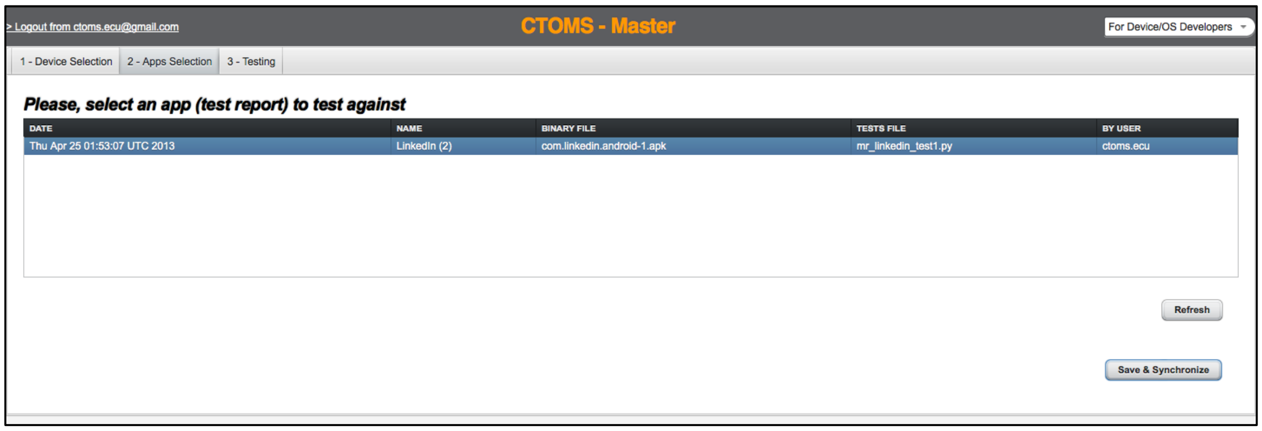
Please, select devices to test on (including private ones from your Node)

DOR	MODEL	ANDROID	RESOLUTION	CONNECTION	PROCESSOR	RAM	NODE	LOCALE	ANDROIDVIEWCLIENT
C	HTC Wildfire S A510e	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256	WindowsNode	ru	☐
awei	U8665	Gingerbread 10 (2.3.3-2	320x480	WiFi	Cortex < 1GHz	256	MacNode	en	☐
nsung	GT-P3113	Jelly Bean 16 (4.1)	1024x600	WiFi	Other	1024	MacNode	EN	☐

☑ Use selected public devices exactly Refresh Rebuild

Save & Synchronize

7) Shows selection of the test session to test against its results (nominal screenshots).


> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For Device/OS Developers

1 - Device Selection | 2 - Apps Selection | 3 - Testing

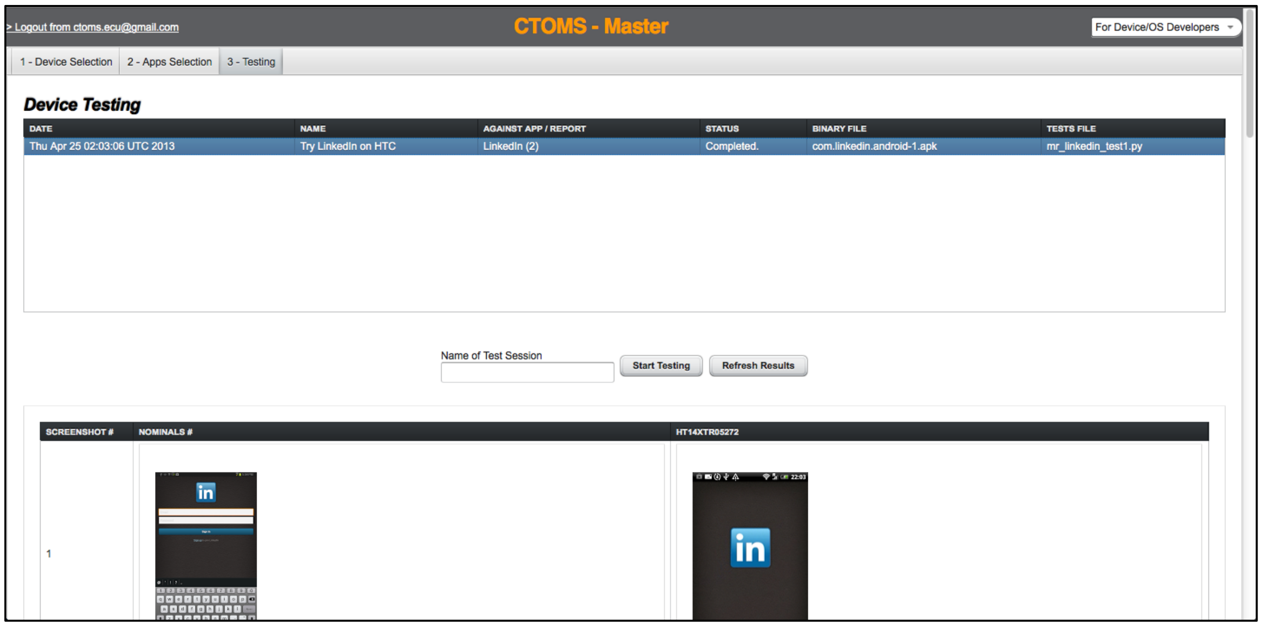
Please, select an app (test report) to test against

DATE	NAME	BINARY FILE	TESTS FILE	BY USER
Thu Apr 25 01:53:07 UTC 2013	LinkedIn (2)	com.linkedin.android-1.apk	mr_linkedin_test1.py	ctoms.ecu

Refresh

Save & Synchronize

8) Shows completed test session of device against app testing.



> Logout from ctoms.ecu@gmail.com

CTOMS - Master

For Device/OS Developers

1 - Device Selection | 2 - Apps Selection | 3 - Testing

Device Testing

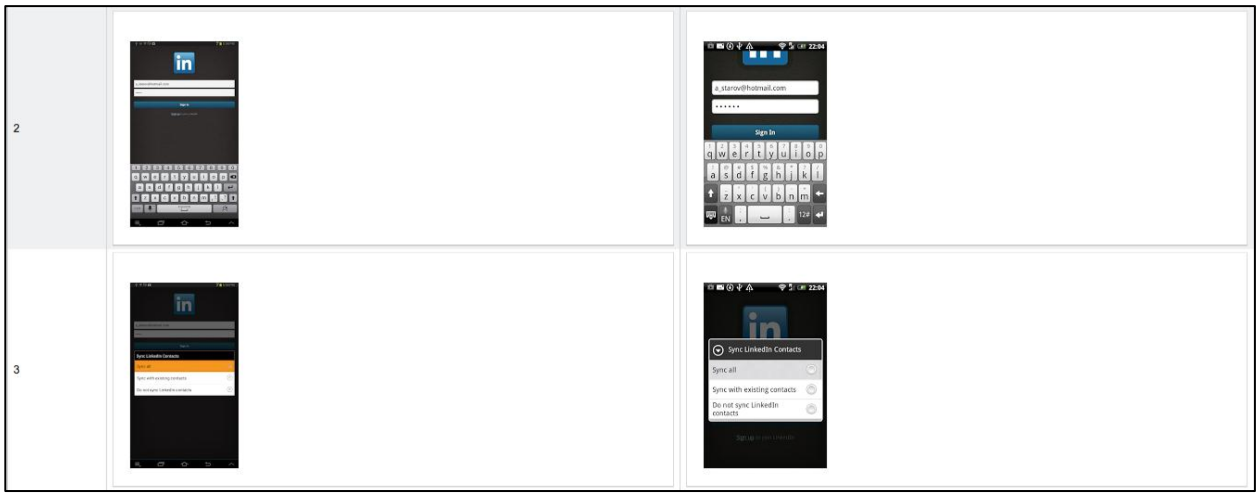
DATE	NAME	AGAINST APP / REPORT	STATUS	BINARY FILE	TESTS FILE
Thu Apr 25 02:03:06 UTC 2013	Try LinkedIn on HTC	LinkedIn (2)	Completed.	com.linkedin.android-1.apk	mr_linkedin_test1.py

Name of Test Session

Start Testing | Refresh Results

SCREENSHOT #	NOMINALS #	HT14XTR05272
1		

9) Shows more results.



2
a_starov@hotmail.com
Sign In
3
Sync LinkedIn Contacts
Sync all
Sync with existing contacts
Do not sync LinkedIn contacts