APPROACHES FOR CONTEXTUALIZATION AND LARGE-SCALE TESTING OF MOBILE APPLICATIONS

A Thesis
Presented to
The Academic Faculty

by

Jiechao Wang

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
August 2013

APPROACHES FOR CONTEXTUALIZATION AND LARGE-SCALE TESTING OF MOBILE APPLICATIONS

Approved by:

Professor Raghupathy Sivakumar, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Ghassan Al-Regib
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Raheem Beyah
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 8 May 2013

To my father Yi Wang,

and

my mother Yuping Wen.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Raghupathy Sivakumar, for his unflagging guidance and support. This thesis would not have been possible without all the insightful discussion with him. During my thesis study, he has acted as an excellent role model of a researcher with intense enthusiasm and drive. His high standards for clear thinking and effective communication will continue to guide me in my future endeavors.

I would like to thank Profs. Ghassan Al-Regib and Raheem Beyah for serving in my thesis committee. I am grateful for their valuable advices and opinions, which helped me improve the quality of this thesis a lot.

My gratitude extends to the present and past members of the Georgia Tech Networks and Mobile Computing Research Group. I thank Sandeep Kakumanu, Cheng-Lin Tsao, Shruti Sanadhya, Chao-Fang Shih, Bhuvana Krishnaswamy and Uma Parthavi Moravapalle for their valuable feedback and assistance in my dissertation.

Last but not the least, I would like to thank my father and my mother. Their encouragement, support, and sacrifices have helped me go through this journey. To them, I dedicate this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

In this thesis, we focused on two problems in mobile application development: contextualization and large-scale testing. We identified the limitations of current contextualization and testing solutions. On one hand, advanced-remote-computing-based mobilization does not provide context awareness to the mobile applications it mobilized, so we presented contextify to provide context awareness to them without rewriting the applications or changing their source code. Evaluation results and user surveys showed that contextify-contextualized applications reduce users' time and effort to complete tasks. On the other hand, current mobile application testing solutions cannot conduct tests at the UI level and in a large-scale manner simultaneously, so we presented and implemented automated cloud computing (ACT) to achieve this goal. Evaluation results showed that ACT can support a large number of users and it is stable, cost-efficiency as well as time-efficiency.

CHAPTER I

MOBILE APPS AND CONTEXT AWARENESS

1.1 Introduction

Mobile applications are very convenient and powerful today. People access their functionalities from anywhere and at any time they want. However, there are still a lot of PC applications that people could only access from their PCs. Therefore, one of the problems in mobile application development is how to make the functionalities of these PC applications also available on mobile devices, i.e. mobilization of PC applications into mobile applications.

There are mainly four approaches to mobilization: building, porting, simple remote computing and advanced remote computing. The limitation of building is that it takes too much time and effort. The limitations of porting are it's time- and effort-consuming and infeasibility. The limitation of simple remote computing is that it will give very bad user experience on mobile devices. The fourth approach, advanced remote computing, don't have any of the aforementioned limitations.

However, because PC applications are not context-sensitive, when advanced remote computing solutions (such as *Mobile) mobilize PC applications, the resulting mobile applications are not context-sensitive either. By context we mean "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [37] But providing context awareness in mobile applications are quite beneficial. Thus we focus on providing context awareness to *Mobile-mobilized applications in this work.

1.2 Problem Statement/Motivation

With the rapid growth in the industry, smartphones have recently hit a long-anticipated milestone: they overtook PCs in terms of the global shipments [31]. The dramatic growth in smartphones, e.g. 87.2% shipment increase in the worldwide market in 2010 [21], demonstrates its promising potentials. Smartphones have occupied more than one quarter of the mobile phone population in US and Europe [15], and they are expected to dominate the mobile market in the near future [32].

Meanwhile, a lot of mobile apps are also created and downloaded for mobile devices, especially from Apple's App Store for iOS devices and Google's Google Play for Android devices. Apple's App Store, the largest digital distribution platform for mobile apps, provides a collection of 550,000 mobile apps. More than 25 billion apps have been downloaded so for in Apple's App Store. Google Play is catching up with 300,000 apps [13].

Since mobile devices almost entirely rely on its mobile applications to provide functionalities to the users, mobile applications are very important to these mobile devices. However, there are many PC applications that don't have counterpart mobile applications on mobile devices. But people still want to access the functionalities of these PC applications from their mobile devices. Therefore, one challenge in developing mobile applications for mobile devices is how to make PC applications available on mobile devices.

To solve this challenge, we need application mobilization to provide functionalities of PC applications on mobile devices. Application mobilization means the process of enabling a PC application to be used from mobile devices [44]. There are four approaches to achieve application mobilization today:

1. Building. People can build the application from scratch and provide some set of functionalities of its PC counterpart on mobile devices. This requires the most effort, because no PC application code is reused. For example, some

social networking sites originally written in web programming languages are now completely rewritten for iOS and Android devices. The limitation of this approach is that the functionalities of a PC application are needed to be completely rewritten for the mobile devices. Sometime this would take another few weeks or months and a lot of effort for each of the mobile platforms that people want to support. Also, after investing a significant amount of time and effort in building the applications, the applications typically expose only a subset of the capabilities of their counterpart PC applications.

2. Porting. People can also port some PC applications to mobile devices. This usually requires less effort than the first approach, since it reuses some of the existing PC application code. One popular example is porting PC applications written in Java to mobile applications written in Android. However, one of the limitations of this approach is that the functionalities of a PC application are required to be partially rewritten for the mobile devices. Moreover, even when the time and effort of this rewriting are affordable, this approach is usually infeasible because of the fact that PC applications and mobile applications are typically written in different programming languages. Take the Java to Android porting example, the development effort could still be non-trivial: developers still need to rewrite all of the Java UI code in Android since the widely-used Java UI libraries Swing and AWT are not supported in Android.

3. Simple remote computing. People could use simple remote computing solutions such as Virtual Network Computing [34] to achieve application mobilization. Simple remote computing solutions allow users to remotely access and control another computer from their computers. Mobile device users can also use simple remote computing solutions to access applications run on PCs. But the views and user interfaces of the PC applications given by simple remote computing

solutions do not fit the small screens of mobile devices, resulting in very bad user experience. Users need to take much more time and effort to complete the same task on their mobile devices than on their PCs.

4. Advanced remote computing. [44] introduces an advanced remote computing solution called *Mobile. At the high level, *Mobile uses a proxy-based network architecture to mobilize PC applications for mobile devices without explicit effort in development [44]. Moreover, its interface properly fits the requirements of a mobile device and it can expose the full functionalities of a PC application. It consists of a backend and a frontend. The backend resides on a PC and the frontend resides on a mobile device. The *Mobile backend offers an ”optimized window” into the backend application for the mobile device user to view and control the application on the *Mobile frontend. This solution does not require explicit development effort and can mobilize PC applications dynamically for multiple mobile platforms.

*Mobile does not have any of the aforementioned limitations. It reuses the PC application logic and does not require rebuilding of the mobile applications. It can achieve application mobilization in a fraction of time and effort of the first and second approaches. It can apply to virtually all PC and web applications. The mobile applications mobilized by it can be run on different mobile platforms (iOS, Android, smartphones and tablets) and guarantee good mobile user experience. Therefore, *Mobile is the most attractive application mobilization approach and we pick it to be studied in our work of contextualization.

Figure 1 illustrates the network architecture of *Mobile. The *Mobile backend runs on a backend PC, which communicates with the *Mobile frontend running on a mobile device. The frontend receives updates to the application views from the backend, and renders those views. The frontend sends any user-input back to the backend, and the backend executes these inputs on the PC application. The *Mobile frontend shown in Figure 1 is a single native application launcher on the mobile device that contains one mobile application to be launched for each application installed on the PC.

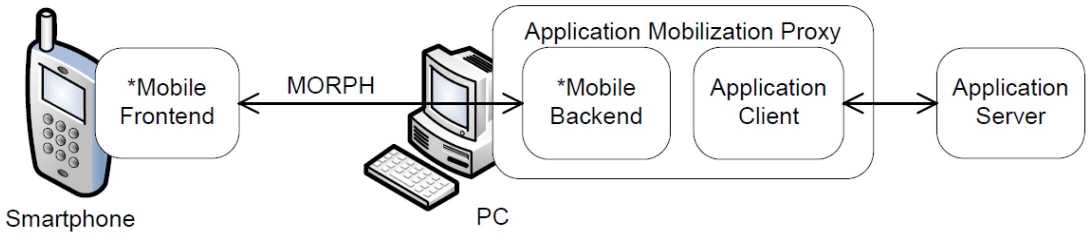


Figure 1: Network Architecture of *Mobile

The problem with *Mobile is that it does not provide context awareness to the applications it mobilizes.

We choose the following context definition [37] to use in this work as it considers the interaction between a user and an application: "context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

The context in PC application is usually unchanged. For example, a PC application is typically always used in the same location as the PC itself does not move. As a result, there is no benefit of building context awareness into a PC application and thus PC applications are not designed to be context sensitive.

But unlike PC application, mobile application is typically used in a changing context. Mobile user might want to know the nearest restaurant and this information changes based on the current location of the mobile device. The user might also want to know the most desirable path for him to drive from one place to the other. This path is time-sensitive as the traffic conditions in different sections of different paths vary with time. It is desirable to provide context awareness in mobile applications so that we can save some of user's time and effort when using it.

Here is an example demonstrating the benefit of providing context awareness into *Mobile solution: suppose that a user is viewing a document when using an application mobilized by *Mobile on his mobile device and wants to print this document. When he issues a print command from his mobile device, there will be a dialog or window popping up asking him to type in the name of the printer or choose one of the printers from a drop-down list. Assuming that other people around him all use a particular printer and *Mobile is aided by context awareness, it would automatically recommend that printer to the user (showing as the first printer in the drop-down list). The user can then issue that print command from his mobile device without taking the time and effort in typing the printer name or making a decision as to which printer to use.

However, the applications mobilized by *Mobile do not come with context awareness, since the PC application itself does not have any context awareness. So one of the problems we are going to solve in this work is to obtain and provide context information in *Mobile-mobilized applications in a seamless fashion to save users' time and effort when they are using them. We will refer to this as the contextualization problem in the following sections. Note that some derivations of our work may also apply to other mobilization approaches, but we only focus on providing context awareness to *Mobile-mobilized applications in this work.

*To summarize, PC applications do not have built-in context awareness. Therefore, if we mobilize these PC applications through advanced remote computing solutions such as *Mobile, the resulting mobile applications don't have context awareness either. So what we want to do in this thesis is overlaying context in those mobile applications to improve user experience and reduce user burden without rewriting the applications or changing their source code.*

1.3 Related Work

There has been significant work in "context-aware computing". Many of them focus on one of the following three topics: definition of context, approaches to context-awareness and applications of context-awareness.

1.3.1 Definition of Context

The works in this category focus on providing a better definition or understanding of context.

For example, [37] discusses some of the research challenges in understanding context and developing context-aware applications. They also give a well-cited definition of context: "context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". We use this context definition in our work since it considers the interaction between a user and an application.

[36] provides an operational definition of context and discuss the different ways that context can be used by context-aware applications. They use the same context definition as in [37]. They also present the Context Toolkit, an architecture that supports common features required by context-aware applications: capture and access of context, storage, distribution, and independent execution from applications. Unlike our work, this architecture focuses on general context-aware applications, not on mobile applications specifically. Moreover, this architecture focus only on physical context, while we focus on both physical context and application state.

[35] defines context, identifies categories of contextual information, and characterizes context-aware application behaviors. The authors present a conceptual framework that separates the acquisition and representation of context from the delivery and reaction to context by a context-aware application.

1.3.2 Approaches to Context-awareness

The works in this category focus on the approaches for computing devices, such as mobile devices, to get context information from environments they are in.

For example, [38] argues that facilitating the interaction between a user and a mobile device requires a personalization of the context awareness. They present that there are two approaches to context awareness: the ontology approach and the unsupervised approach. They argue that the latter one is more workable compared to the former one.

[45] presents a prototype implementation for the detection of movement patterns based on the built-in sensors of an Android smartphone, which processes sensor information in a neural network to match them to certain contexts of a mobile user. Unlike our work that focuses on both acquiring and using different kinds of context, this work only focuses on acquiring of movement patterns.

[41] proposes a new way to collect context in situations with many devices, such as mobile phones. Unlike our work that focuses on providing context awareness to mobile applications, this work focuses on the determination of context by multiple mobile devices when they are close to each other.

1.3.3 Applications of Context-awareness

The works in this category focus on presenting a variety of applications to demonstrate the usefulness of context-awareness.

For example, [40] presents a quality-of-serviceaware decision engine for content adaptation service of mobile devices. This work focuses on building a good content adaptation service for mobile devices so that users can access the variety of rich web hypermedia content from their mobile devices.

[43] describes ConChat, a context-aware chat program that enriches electronic communication by providing contextual information and resolving potential semantic

conflicts between users.

[42] proposes two systems in a computer supported ubiquitous environment for language learning. The first one is a support system for Japanese polite expressions learning. The second one detects the objects around learner using RFID tags and provides the learner the educational information.

[46] presents a Kimura system that augments and integrates independent tools into a pervasive computing system and monitors a user's interactions with the computer, an electronic whiteboard, and a variety of networked peripheral devices and data sources. They seek to design an office that better supports knowledge workers: business professionals who interpret and transform information.

CHAPTER II

CONTEXTIFY: AN OVERLAY APPROACH TO CONTEXTUALIZING MOBILE APPS

2.1 Solution Design

2.1.1 Problem Restatement

Mobile applications are very powerful and convenient. People access their functionalities from anywhere and at any time they want. However, there are still a lot of PC applications that people could only access from their PCs. Therefore, one of the problems in mobile application development is how to make the functionalities of these PC applications also available on mobile devices, i.e. mobilization of PC applications into mobile applications.

There are mainly four approaches to mobilization: building, porting, simple remote computing and advanced remote computing. The limitation of building is that it takes too much time and effort. The limitations of porting are it's time- and effort-consuming and infeasibility. The limitation of simple remote computing is that it will give very bad user experience on mobile devices. The fourth approach, advanced remote computing, don't have any of the aforementioned limitations.

However, because PC applications are not context-sensitive, when advanced remote computing solutions (such as *Mobile) mobilize PC applications, the resulting mobile applications are not context-sensitive either. But providing context awareness in mobile applications are quite beneficial. Thus we focus on providing context awareness to *Mobile-mobilized applications in this work.

2.1.2 Solution Overview

In this section, we introduce the overview of contextify. When users are required to give inputs, our solution first records users' input actions and the context where these input actions happen in the mobile applications. Then our solution learns or configures the relationship between the context and these input actions. Using this relationship, our solution can then provide overlaid suggestions for users' input actions through some UI elements in the mobile application. This is done by using any of the three kinds of contexts introduced in the next section.

2.1.3 Input Actions and Context

We define users' input actions as any clicks, scrolls, touches or keyboard presses that user performs on UI elements in the mobile application. Four things are recorded in the mobile application that we want to provide context awareness to: the types and sequences of these input actions, the time when these input actions happen and the context in which these input actions happen.

Three kinds of mobile-application-related contexts are utilized in contextify:

1. What this user did before.
2. What this user did on PC.
3. What other users did before.

Recall that the context definition we use in this work is "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [37]. All of the above three kinds of contexts fall under this definition.

Once the related input actions and contexts are recorded, we need to establish the relationship between these input actions and these contexts in order to give recommendations for users' input actions later in the given context. This relationship can be established in one of the two ways:

1. It can be learnt automatically if there are previous users who give input actions in the same contexts. In this case, we can predict that the new user will probably perform the same input actions if the mobile application is in the same context. These input actions tell us what task the new user will probably perform, so that we can recommend the new user to do the same task through some new UI elements added to the mobile application. If the new user does intend to perform the recommended task, then his/her time and effort are saved.

2. In cases we cannot get previous users' input actions in the same context, the relationship between the context and the input actions cannot be learnt automatically. So we configure these relationships in contextify. After the relationships are established in this way, when the mobile application is in one of the recorded contexts, we can map the current context to a sequence of input actions and recommend the user to do the task accomplished by these input actions. Similar to the above case, if the user does intend to perform the recommended task, then his/her time and effort are saved.

2.1.4 Design Elements

In this section we present the three design elements of contextify:

1. Frontend UI elements. They are added to the *Mobile-mobilized applications for providing context awareness. These UI elements are used to recommend input actions to the users and allow them to perform the predicted task in a more time-efficient and effort-efficient way. Normally these UI elements are a page or a dialog containing a list of recommended items. If users find these

recommendation lists useless, they can easily dismiss them with only one additional click. In this way, contextify only adds very small overhead to users' time and effort when its recommendations are not useful.

2. A online file storage service. We use it to provide file information to our system. Instead of editing files on mobile devices, it is much likely for users to edit these files on PCs. But users may want to upload and view these files on their mobile devices wherever they go and at any time they want, so this service or at least the file information provided by this service should be accessible from both PCs and different mobile devices.

3. A backend database. The third design element in contextify is a database at the *Mobile backend to maintain and provide information for recommendations. It will record and maintain context information such as which files have been viewed and how many times each of these file has been viewed. We place this database at the backend so that the information it maintains can be used in all *Mobile-mobilized applications across all mobile devices.

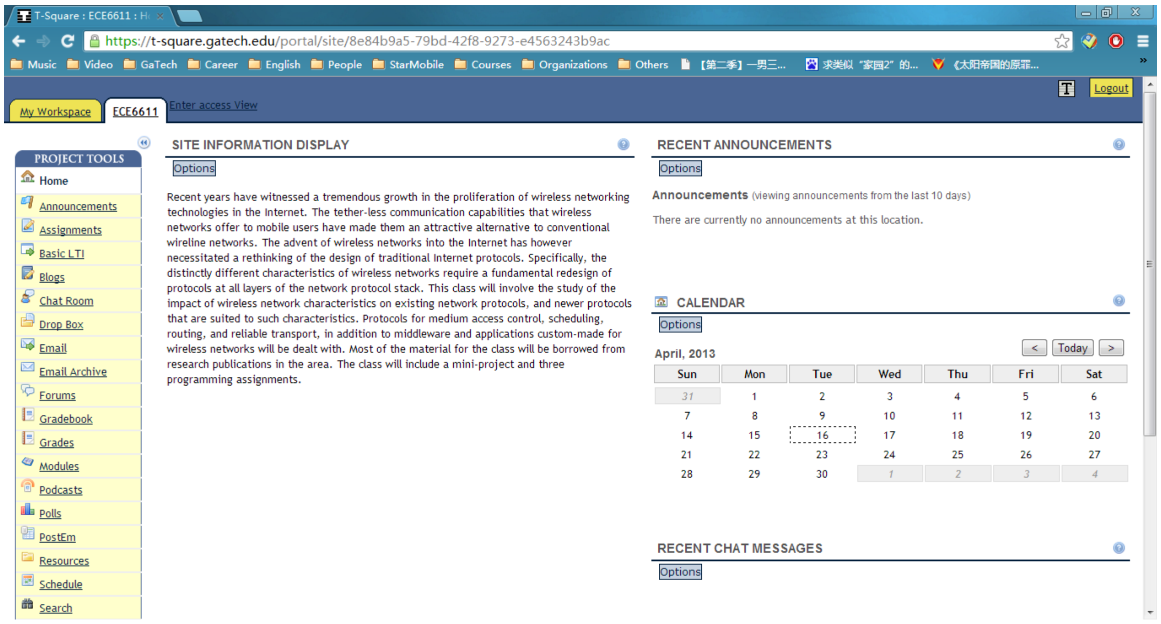
2.2 Solution Implementation

2.2.1 T-Square

In this section, we introduce the implementation of contextify. We first introduce the *Mobile-mobilized application that we want to provide context awareness to and two of its use cases. Then we introduce the system architecture of contextify. contextify is a general solution that applies to all *Mobile-mobilized applications. But in this work, we only implement it in the following application.

T-Square is a Collaboration and Learning Environment (CLE) based on the code produced by the Sakai open source community, of which Georgia Tech is a leading member [33]. Its user can be either a Georgia Tech instructor or a Georgia Tech student. It has a site for each of the courses taught by an instructor or taken by a student, and inside each site, a set of tools is provided to the instructor or the student to use for that course, such as the Resources tool to upload, view and download files, the Announcements tool for instructors to make announcements and for students to view them, the Wiki tool for instructors and students to collaboratively add, edit and read information about that course, etc. It is a popular PC application among Georgia Tech instructors and students. It has also been mobilized by *Mobile. So we choose to base our implementation on it in this work.

Please see Figure 2 for a screenshot of T-Square website running in a Chrome browser on a PC.



Recent years have witnessed a tremendous growth in the proliferation of wireless networking technologies in the Internet. The tether-less communication capabilities that wireless networks offer to mobile users have made them an attractive alternative to conventional wireline networks. The advent of wireless networks into the Internet has however necessitated a rethinking of the design of traditional Internet protocols. Specifically, the distinctly different characteristics of wireless networks require a fundamental redesign of protocols at all layers of the network protocol stack. This class will involve the study of the impact of wireless network characteristics on existing network protocols, and newer protocols that are suited to such characteristics. Protocols for medium access control, scheduling, routing, and reliable transport, in addition to middleware and applications custom-made for wireless networks will be dealt with. Most of the material for the class will be borrowed from research publications in the area. The class will include a mini-project and three programming assignments.

Announcements (viewing announcements from the last 10 days)

There are currently no announcements at this location.

April, 2013

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4

Figure 2: T-Square website on PC

2.2.2 Use Case 1

We've identified and implemented two use cases in T-Square that can be benefited from context awareness. We introduce use case 1 in this section.

Use case 1: a student recently edits some files in his Dropbox folder on the PC. When he goes to the upload page in the Resources tool from a mobile device, he will be prompted with a dialog asking him which of these files he wants to upload. He can then choose one of the files by clicking on the corresponding button or dismisses the dialog if he doesn't want to upload any of these files. The context here is the upload page in the Resources tool and the users' input actions are the click and scroll actions that are needed to upload one of his recently-edited files. This is the case where there is no way to learn the relationship between the context and the input actions, so we configure the relationship here in contextify.

We now introduce the implementation of this use case. We add code to the *Mobile-mobilized application frontend so that we can know when the user enters the upload page in the Resources tool. When he does so, contextify uses Dropbox API to tell us which files in this users' Dropbox folder are edited recently and prompt a list of these files in a dialog to the user. The user can then choose to dismiss the dialog or click on a button representing one of the files. If the user clicks any such button, we use Dropbox API to download the selected file from his Dropbox folder to his mobile device, and then he can upload this file from his mobile device to the Resources tool, without the need to browse his local file system and locate the file he wants to upload.

Screenshots for this use case are shown in Figure 3, Figure 4 and Figure 5. Figure 3 shows the upload page in the Resources page where users can click on the "Choose File" button and select a file to upload. After a user clicks on that button, contextify shows a list of files that the user recently edits in his Dropbox folder on the PC, as shown in Figure 4. Figure 5 shows that the "Display Name" field has been updated with the name of the file chosen by the user.

2.2.3 Use Case 2

We introduce use case 2 in this section.

Use case 2: some Georgia Tech students have recently downloaded some files. When another Georgia Tech student goes to the Resources tool from a mobile device,

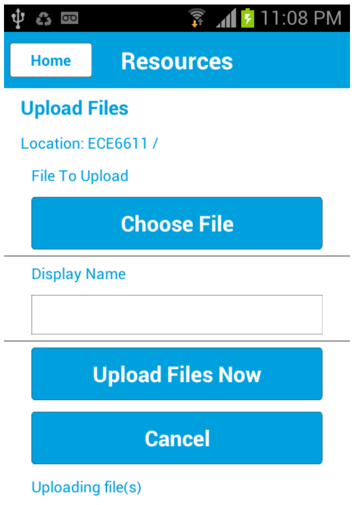


Figure 3: Upload page on mobile device

there will be a dialog asking him whether he wants to download and open any of those files. He can then download and open one of these files by clicking on the corresponding button. The context here is the Resource tool and the input actions are those click and scroll actions that are needed for other Georgia Tech students to download these files. Contrast to use case 1, this is the case where the relationship between the context and the input actions can be learnt automatically because the other users perform these input actions in this context. From these input actions, contextify learns that the users were trying to download a set of files. Both these input actions and the context are recorded in contextify so that whenever the mobile application is in the same context again, contextify can recommend the same set of files for the user to download. If the user indeed wants to download a file from the recommended set, he can directly click on the corresponding button, without the need to locate these files in the Resources tool.

We now introduce the implementation of this use case. We add code to the *Mobile-mobilized application frontend so that we know whenever a user enters the Resources tool. These users may download and open some files in the Resource tool.

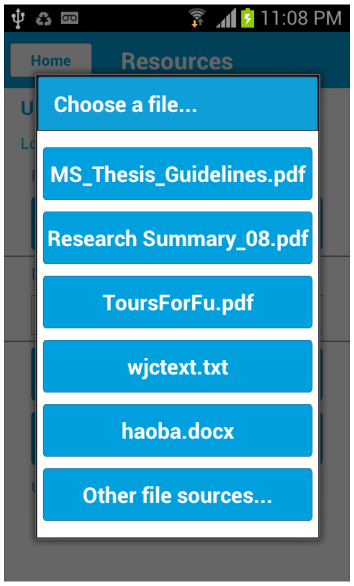
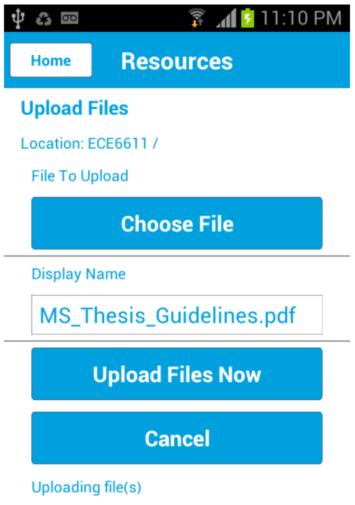


Figure 4: The list of recently-edited files recommended by contextify

If they do so, contextify at the application frontend will record the names of these files and send this information to the application backend. At the application backend, we maintain the names of the files viewed by users in the Resources tool and how many times each of these files is viewed. By utilizing this information, whenever the mobile application is in the Resource tool, the top ten most viewed files are recommended to the user as a list of buttons in a dialog.

Screenshots for this use case are shown in Figure 6, Figure 7, Figure 8 and Figure 9. Figure 6 shows the page before the Resources tool. After the user clicks on the ”Resources” button, contextify prompts him with a page containing a list of files. Those files are recently viewed by all the users of this site and are recommended to the user to open. After the users selects a button, the corresponding file is opened with a file reader program on the device, as shown in Figure 8. After the user finishes viewing the file, he can go back to the T-Square Resources tool by clicking on the hardware back button, as shown in Figure 9.



Location: ECE6611 /

File To Upload

Display Name

MS_Thesis_Guidelines.pdf

Uploading file(s)

Figure 5: After user selects one of the recently-edited files to upload

2.2.4 System Architecture

The system architecture includes three parts: an Android add-on to the *Mobile-mobilized application frontend for detecting and collecting context information and providing user interface for users to interact with the whole system; A Dropbox file storage and sharing component to allow users to edit files on their PCs and provide information of these files to the other parts of this system; a database at the *Mobile-mobilized application backend server to record names of the files viewed by users and how many times they are viewed. The overall architecture is shown in Figure 10

2.2.5 Other Use Cases

In this section, we introduce three other use cases and their implementations. They are considered but not implemented in this work.

Use case 1: suppose that a user is viewing a document in a *Mobile-mobilized application on his mobile device and would like to print this document. When he issues a print command from his mobile device, he will see a dialog popping up asking him to type in the name of the printer or choose one of the printers from a

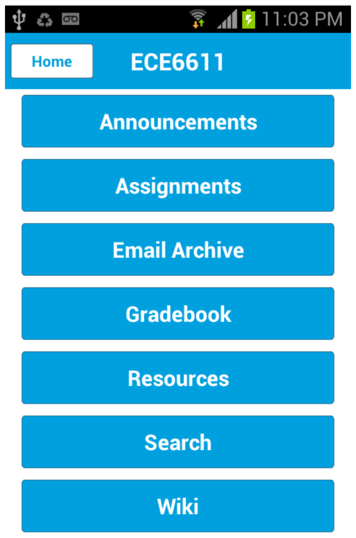


Figure 6: Before user enters the Resources tool

drop-down list. Assuming that other people around him all use a particular printer, it would automatically recommend that printer to the user (showing as the default value and the first printer in the drop-down list). The user can then issue the print command from his mobile device without taking the time and effort to enter the printer name or making a decision as to which printer to use.

We now introduce the implementation of this use case. We maintain a database at the *Mobile backend to record related information when any user issues a print command. Each entry in the database contains four elements: the name of the Wi-Fi network the device is connecting to, the location of the mobile device, the printer name and the number of times this printer has been used. Note that it is possible that more than one printers have been used in the same Wi-Fi network and at the same location, so we also record all these printers and the number of times each printer is used in the database. When a new user issues a print command, we will only suggest the most-used printer with the matching Wi-Fi and location. If there is no printer in the database that has both matching Wi-Fi and location, we will suggest the printer either with matching Wi-Fi only or matching location only.

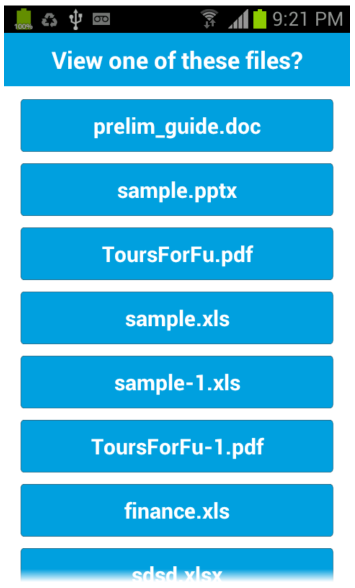
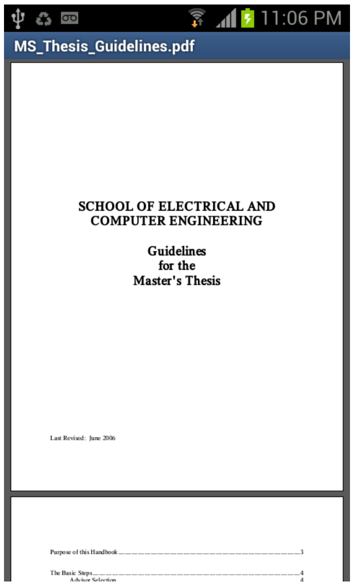


Figure 7: A list of recently-viewed files recommended by contextify in the order of number of view

Use case 2: depending on users' moving speeds, we reduce the number of UI elements they can see on *Mobile-mobilized applications. We have three levels of UI elements richness according to users' three moving states: static, walking and jogging. When a user is in static state, no UI elements will be reduced from *Mobile-mobilized applications to ensure his access to full functionalities. When he is in walking state, we reduce some of the UI elements and only show him the UI elements that he will likely want to see on the mobile device. When he is in jogging state, we will only show a smallest set of UI elements to him. This set contains only the UI elements that he may need to access when he is moving at this fast speed.

We now introduce the implementation of this use case. The walking speed of a human being is 2.5 to 4 miles an hour and the average jogging speed is anywhere between 4 to 5.5 mph [28], so we use 2.5 as the threshold to distinguish between static and walking and 4 to distinguish between walking and jogging. Any speed lower than 2.5 is considered as static and any speed higher than 4 is considered as jogging. We use Android location API [11] to determine the moving speed of mobile devices and use that information in *Mobile-mobilized applications. We use *Mobile's zoom service


11:06 PM

MS_Thesis_Guidelines.pdf

SCHOOL OF ELECTRICAL AND
COMPUTER ENGINEERING

Guidelines
for the
Master's Thesis

Last Revised: June 2006

Purpose of this Handbook 3

The Basic Steps..4

Figure 8: User clicks on a button

to dynamically reduce the UI elements in *Mobile-mobilized applications. The zoom service allows a user to adjust richness of the frontend with several zoom levels. The lowest zoom level provides the simplest user interface with the most limited set of features, and the highest zoom level enables all features that the user wants to access from a mobile device [44]. We also maintain a database at the *Mobile backend to record how many times each UI elements on each page is used by each user in each state.

Use case 3: assuming that two people are meeting at a place such as Starbucks. One of them is using a *Mobile-mobilized email client on his mobile device and would like to search for the emails sent from or to the other person. He goes to the search page in the email client and sees the text field to type in the name of the other person. Figure 11 shows the search page in the Gmail app. The *Mobile-mobilized email client should be able to suggest the name or the email address of the other person he is meeting with by providing it as a default value or the first item in the drop-down suggestion list.

We now introduce the implementation of this use case. Since the two people in

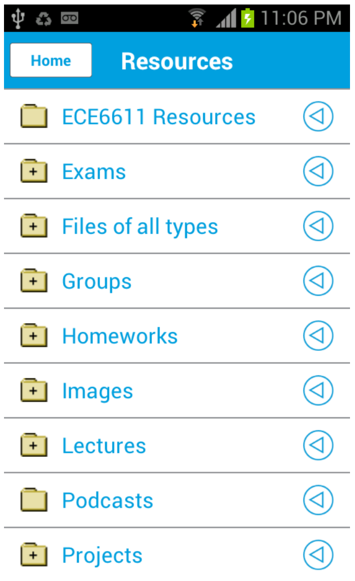


Figure 9: The page for the Resources tool

meeting are sitting or standing close to each other, their mobile devices are also close to each other. So we use Android Bluetooth API [8] to get related information about the other person's mobile device. Android users often associate their Gmail accounts with their mobile devices so that we can get the Gmail address of the other person in the meeting. We can then provide this email address as a default value or the first item in the suggestion list of the *Mobile-mobilized email client search text field where the user is expected to type in the name of the other person.

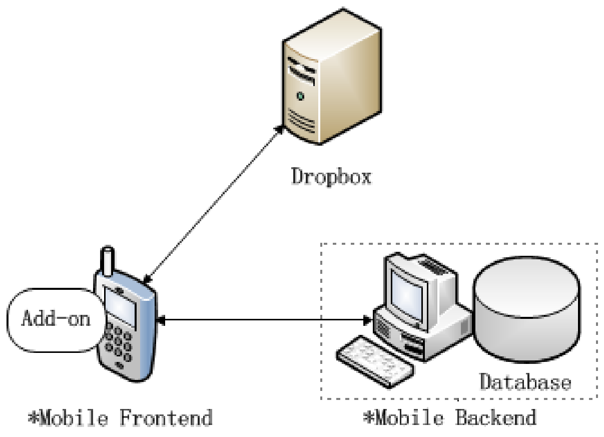
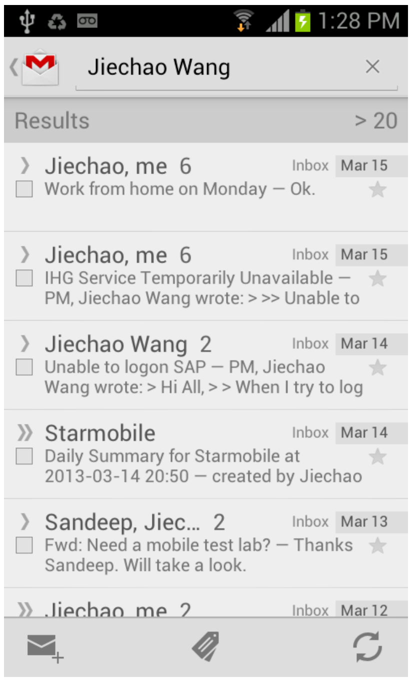


Figure 10: The overall architecture of contextify



1:28 PM

Jiechao Wang

Results > 20

Jiechao, me 6 Inbox Mar 15
Work from home on Monday — Ok.

Jiechao, me 6 Inbox Mar 15
IHG Service Temporarily Unavailable — PM, Jiechao Wang wrote: > >> Unable to

Jiechao Wang 2 Inbox Mar 14
Unable to logon SAP — PM, Jiechao Wang wrote: > Hi All, > > When I try to log

Starmobile Inbox Mar 14
Daily Summary for Starmobile at 2013-03-14 20:50 — created by Jiechao

Sandeep, Jiec... 2 Inbox Mar 13
Fwd: Need a mobile test lab? — Thanks Sandeep. Will take a look.

Jiechao, me 2 Inbox Mar 12

Figure 11: Search for the emails related to a person

CHAPTER III

PERFORMANCE EVALUATION OF CONTEXTIFY

In this chapter, we introduce the evaluation criteria and results for contextify. We use both quantitative and qualitative criteria to evaluate contextify.

3.1 Quantitative

We have designed two scenarios that our recommendations in the two implemented use cases are useful in one scenario and non-useful in the other scenario. For use case 1, when a user goes to the upload page in the Resources tool, he is prompted with a list of recently-edited files to upload. If he indeed wants to upload one of these recommended files, he can click on the corresponding button and the recommendation is useful in this case; otherwise, if he wants to upload a file other than the files in the recommended list, he needs to dismiss the list and locate that file. In this case, the recommendation is not useful. For use case 2, when a user goes to the Resources tool, he is prompted with a list of files opened by all users. If he indeed wants to open one of these recommended files, he can clicks on the corresponding button and the recommendation is useful in this case; otherwise, if he wants to open a file not recommended here, he needs to dismiss this list and enter the target folder in the Resources tool to open the target file. The recommendation is not useful in this case.

We ask ten volunteers to perform the two implemented use cases in contextualized T-Square application, under both useful scenario and non-useful scenario. We ask them to perform the same two use cases in normal T-Square application as well. In order to eliminate the influence of learning curve that volunteers tend to take less time and make less errors when doing the same use case in the second application, we alternate the order of the two applications each volunteer performs use cases in:

the first volunteer would perform use cases in contextualized T-Square application first and then in normal T-Square application; the second volunteer would perform use cases in normal T-Square application first and then in contextualized T-Square application; the third volunteer would perform use cases in contextualized T-Square application first and then in normal T-Square application, etc.

The ten volunteers involved in this evaluation experiment are all Georgia Tech students. They include both experienced smartphone users as well as new smartphone users. Some of them are technical persons, some of them are not. Throughout the whole experiment, we record both the time and the number of input actions needed for each volunteer to complete each use case in each scenario in each application.

The results are shown in Figure 12 through Figure 15. Compared to normal T-Square, we observe from these four figures that users' time and effort (measured in number of actions) in performing both use cases in contextified T-Square are reduced significantly under useful scenario, but increased only slightly under non-useful scenario. Since users need to dismiss the recommendation lists in both use cases under non-useful scenario, the slightly increase of users' time and effort are expected. The reduce in users' time and effort under useful scenario are much larger than the overhead occurred under non-useful scenario, demonstrating the effectiveness of contextify in saving users' time and effort.

3.2 Qualitative

After each volunteer completes his experiment, we ask him to rate his user experience on using both contextualized T-Square application and normal T-Square application on the scale of 1 to 5, with 1 representing the worst user experience and 5 representing the best user experience. This is called the Likert scale [39] and is the most widely used scale. The average rating of normal T-Square given by these ten volunteers is 3.3, while that of contextified T-Square under both useful and non-useful scenarios is

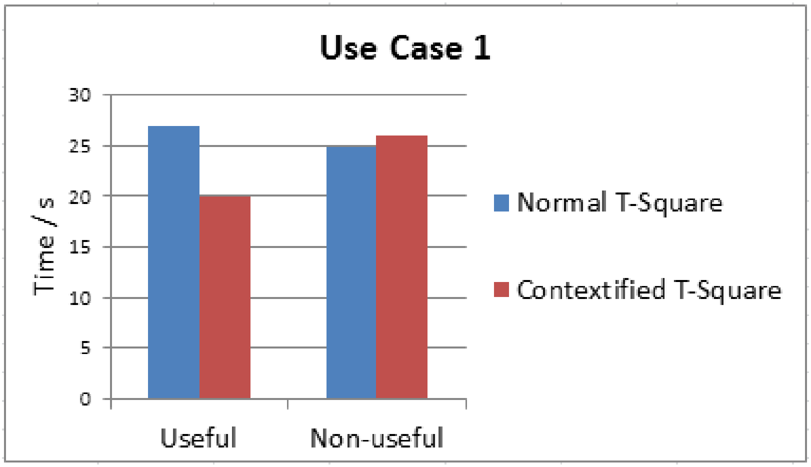


Figure 12: Time to complete use case 1 in normal T-Square and contextified T-Square

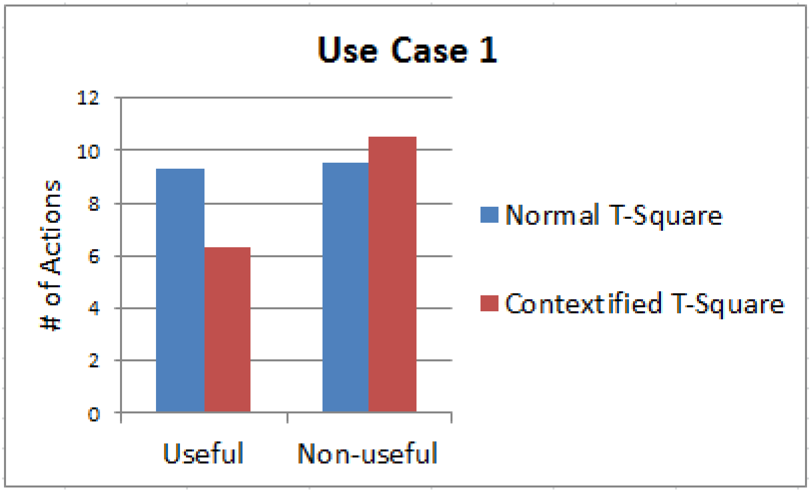


Figure 13: Number of actions to complete use case 1 in normal T-Square and contextified T-Square

4, demonstrating the effectiveness of contextify in bettering user experience.

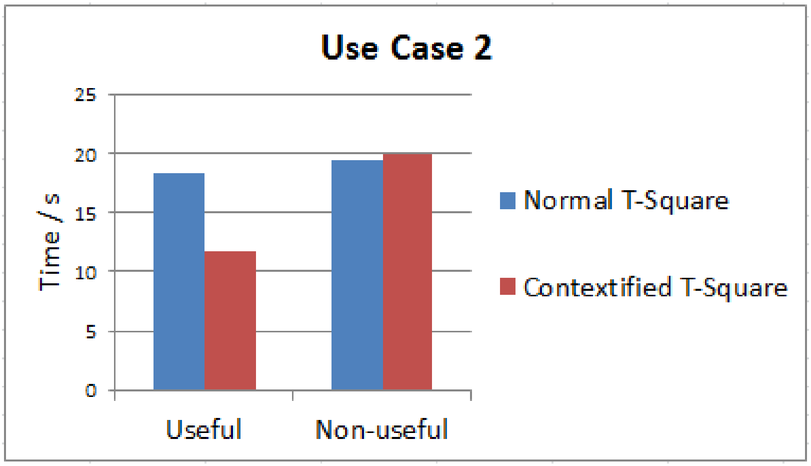


Figure 14: Time to complete use case 2 in normal T-Square and contextified T-Square

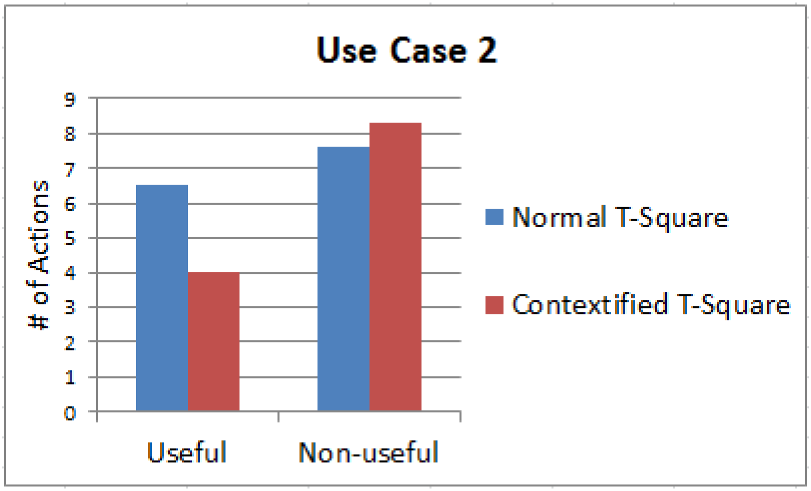


Figure 15: Number of actions to complete use case 2 in normal T-Square and contextified T-Square

CHAPTER IV

LARGE-SCALE TESTING OF MOBILE APPS

4.1 Introduction

Many popular mobile applications today are used by a large number of concurrent users. Hence it is important for mobile application testers to know if a mobile application can support this number of concurrent users without compromising its basic functionalities, performance and stability. This can be done by conducting different types of mobile application testings in a large-scale manner. By "a large-scale manner", we mean running a large number of instances of the same mobile application simultaneously during the testing.

On the other hand, in order to truly mimic the user interactions and provide information about user experience with the concerned mobile application, it is beneficial to conduct mobile application testings at the UI level, instead of recording and then relaying the network traffic between the mobile application's clients and servers.

However, there is no solution to conduct mobile application testings in the large-scale manner and at the UI level at the same time. The mobile testing solutions available today can only achieve one of the above two aspects. Therefore, in this work, we present a solution to enable testers to conduct a variety of mobile application testings in the large-scale manner and at the UI level at the same time.

4.2 Problem Statement/Motivation

In this section, we first present the motivation of this work. We then give two reasons for why do we choose to only focus on testing mobile applications that follow client-server model.

4.2.1 Motivation

As of March 2013, there are about 6 billions mobile subscriptions [16]. Popular mobile applications are used by a large number of users. Sometimes these users even use the same mobile application simultaneously. Therefore, to be able to figure out if a mobile application can support multiple concurrent users and how many concurrent users it can support are very important. These can be done by various kinds of mobile application testings (such as functional testing to validate a mobile application by checking its basic functionalities, performance testing to see how a mobile application performs in terms of responsiveness and stability, and stress testing to test the stability of a mobile application when it is used beyond normal operational capacity) in a large-scale manner. By a large-scale manner, we mean running a large number of instances of the same mobile application at the same time during the testing. We will call ”testing of mobile application in a large-scale manner” as ”large-scale testing” in the following sections.

On the other hand, testing mobile applications through interacting with their UIs is also important, since this is how users actually use these mobile application. Moreover, a testing solution can use UI interaction to tell how the user experience on these mobile applications is. We will call mobile applications testing through UI interaction as ”at the UI level” in this work.

However, there is no standard way of achieving the above two aspects at the same time, i.e., no standard solution for conducting large-scale testing of mobile applications at the UI level. The currently available mobile application testing solutions are either only able to test a few instances of a mobile application at one time, or cannot support mobile application testing at the UI level. For example, for all the large-scale testing or load testing solutions we have surveyed, CloudTest [14], NeoLoad [24] and Agileload [30] only simulate traffics generated by users' interactions with UI elements, whereas Contus [26] and uTest [1] only provide load testing as a service.

So the motivation of our work is to provide a UI-level large-scale testing solution for mobile applications.

4.2.2 Focus Only on Client-Server model

Our solution focuses only on mobile applications that follow client-server model. A mobile application that follows this model has both a client part and a server part. The client part is on the mobile devices (iOS or Android, smartphones or tablets) which contacts its server to request resources (typically data and computational capacity). The server part of the application typically resides in the cloud and shares resources with potentially a large number of clients. One of the reasons that we only focus on client-server model mobile applications is that this model is quite popular with mobile applications, since mobile devices don't have enough computational capacity to do the computation or storage to store all the data so that these tasks are done by the server part of mobile applications. We have conducted survey on how popular this model is in Apple App Store and Google Play. Results show that six out of the top ten free iOS apps and eight out of the top ten free Android apps for smartphones follow this model.

The other reason that we focus only on this model is, mobile apps that do not follow this model don't need large-scale testing. Since they only have a client part on mobile devices for users to use (a local games on mobile devices for example), there is no central place and potential bottleneck in supporting a large number of clients. Because of that, there is no problem with these mobile applications to serve a large amount of users and hence there is no point in conducting various kinds of testings in a large-scale manner.

CHAPTER V

ACT: AUTOMATED CLOUD TESTING PLATFORM FOR MOBILE APPS

5.1 Solution Design

We introduce our large-scale testing solution in this section. We will first introduce the design principle and then introduce the three design elements of our solution.

5.1.1 Design Principle

We want to design a solution to enable the conduction of a variety of mobile application testings at the UI level and in a large-scale manner simultaneously. Conducting the testing at the UI level can truly mimic users' interactions and provide information about user experience on the concerned mobile application. On the other hand, conducting mobile application testing, such as functional testing, performance testing, stress testing, etc., in a large-scale manner can tell whether the concerned mobile application can support many concurrent users without compromising its basic functionalities, performance and stability.

5.1.2 Design Elements

Based on the above design principle, we have three design elements in this solution:

1. Emulation: We use emulators instead of real mobile devices in our testing solution, since it is infeasible to obtain a large number of real mobile devices. Furthermore, unlike other testing solutions that simply record and then play back the network traffics between the clients and the servers, emulators allow us to interact with the application in the UI level.

2. Automation: Since we cannot manually launch and then interact with so many clients, automation is also necessary if we want to support a large number of users. In our solution, after we make the necessary configuration, we can automate the following three processes: creating and launching enough computing instances in the cloud, launching a large number of emulators in these computing instances and launching and interacting with application clients in these emulators.

3. Cloud: We use the cloud to quickly obtain enough computing instances so that we can run a large number of clients of a particular mobile application in these instances. In this way, functional testing, performance testing, stress testing and other kinds of testings can be conducted in a large-scale manner.

Because of these design elements, we will call our solution "automated cloud testing" (ACT) in the following sections.

5.2 Solution Implementation

In this section, we first introduce the architecture of ACT. Then we introduce the steps to prepare and conduct the testing and some additional features we've built into the testing.

5.2.1 Architecture

This section introduces the system architecture in terms of the implementation of the three design elements. Figure 16 shows the overall architecture.

1. For the emulation, we use Android emulator [10] to run the Android version of *Mobile (which is introduced in the previous chapter) frontend in this testing. Figure 17 shows an Android emulator with *Mobile icon on the right bottom corner. The *Mobile frontend is an application launcher for launching the mobilized mobile applications. Once a mobile application is launched, users will

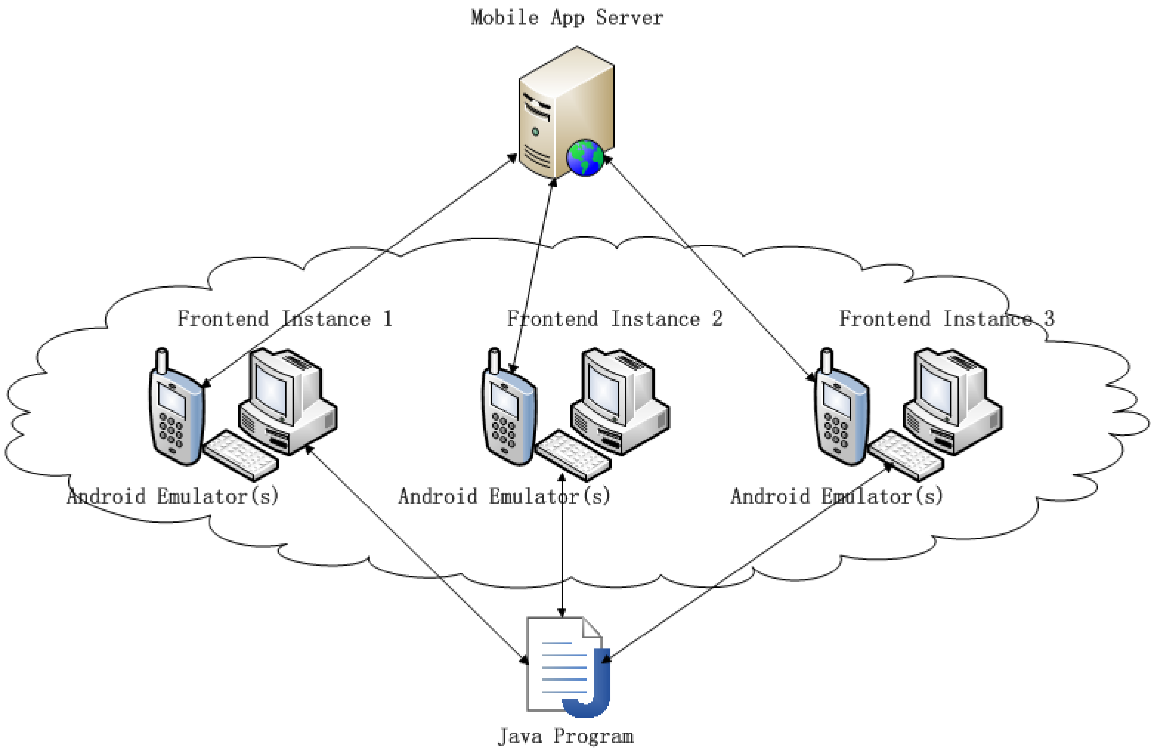


Figure 16: Architecture of the Large-Scale Testing Solution

see the mobile application frontend on their devices. Users interact with the mobile application through user interface in this mobile application frontend.

2. For the automation, we configure automatic logon for each computing instance that Android emulators will run on. Then we use three kinds of script files to launch Android emulators and monkeyrunner [27] instances. Monkeyrunner is a tool that provides an API for writing programs that control an Android device or emulator from outside of Android code [27]. We use monkeyrunner to automatically launch, login and do randomized testing in the mobile application launched in *Mobile

3. For the cloud, we use Amazon Elastic Compute Cloud (Amazon EC2), which is a web service that provides resizable compute capacity in the cloud [6]. We use Amazon Web Services (AWS) Java API [23] in a standalone Java program running on a PC to launch, start, stop and terminate a large number of EC2

instances. These EC2 instances are used to launch Android emulators, launch the mobilized mobile application and the monkeyrunner instances. We call these EC2 instances ”frontend instances” in the following sections, as the mobile application frontends reside on these instances.

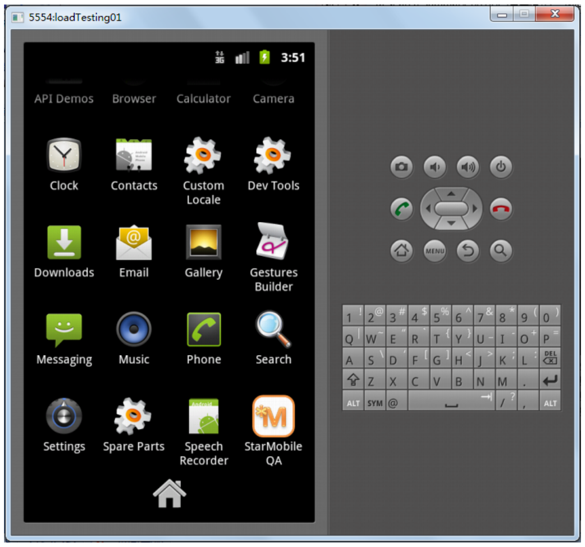


Figure 17: Android Emulator with *Mobile Icon

5.2.2 Test Preparation

In this section, we introduce the steps to prepare ACT. Figure 18 shows the steps to prepare for and conduct the test.

5.2.2.1 Write script files used for test automation

We have created three kinds of script files for automating the test. The first one is a Windows batch file that is used to download all the other necessary script files and execute one of them. We call it downloader file and put it in the Startup folder of a frontend instance in order to execute it automatically once the OS is ready. The second one is also a Windows batch file that is used to launch a specified number of Android emulators (equal to the number of users we want to support in this frontend instance) and their corresponding monkeyrunner instances. We call it launcher file

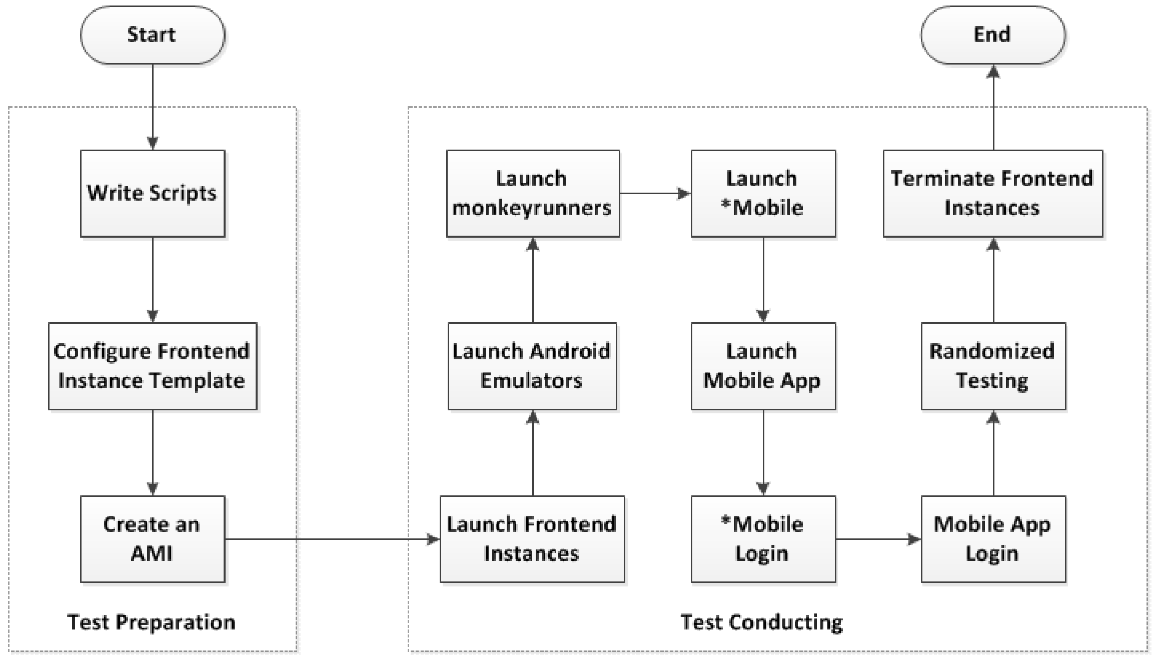


Figure 18: Steps for Test Preparation and Conducting

and put it in the Public folder of our Dropbox account so that it can be downloaded via a URL. This file is the file executed by the downloader file. The third kind of script files are a number of Python files also in the same Dropbox folder. Each Python file is used by one monkeyrunner instance to control an Android emulator.

The reason that we want to include a downloader file only used for downloading all the other files is, we want to put any file that we may need to change later outside of frontend instances so that when these files are changed, we don't need to create a new image to make this change effective to all the frontend instances used in testing. The content of the downloader file is not going to change, because we will always need to download these files and execute the launcher file. But the contents of the launcher file and Python files might change. For example, we might want to support a different number of users in a particular round of testing so that we need to change the launcher file. We might also want to change the duration of randomized testing (explain later) so that we need to change all the Python files.

5.2.2.2 Launch and configure a frontend instance and create an Amazon Machine Image from it

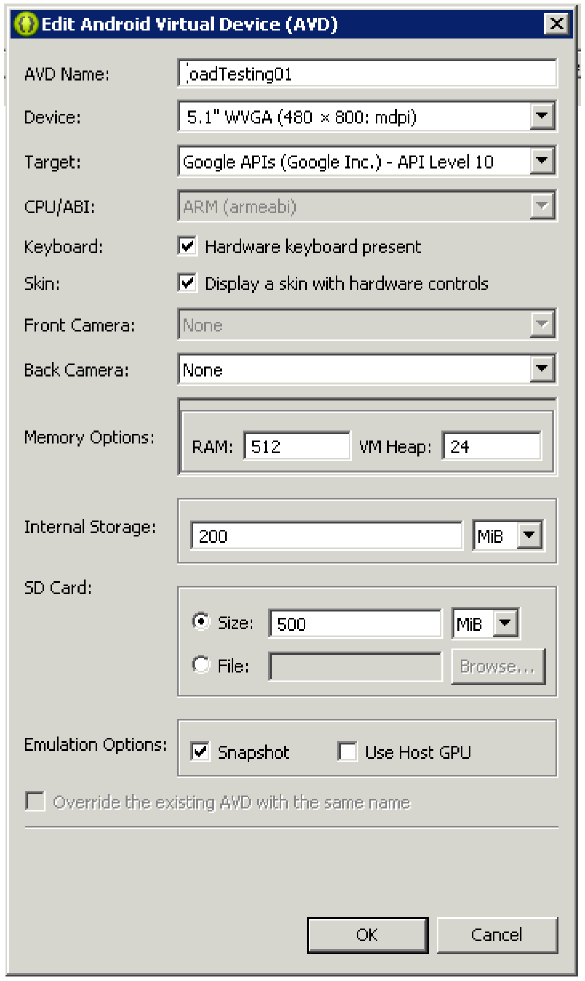
We first launch a frontend instance through AWS control console [3] as a template for all the frontend instances used in conducting the large-scale testing. We use Microsoft Windows Server 2008 R2 Base image provided by Amazon for this instance. After it is ready, we put the downloader file in its Startup folder. Then we configure the automatic logon [20] and create 16 Android Virtual Device (AVD) [25] for this instance. We can only launch up to 16 Android emulators on each frontend instance with these 16 AVDs. This is the maximum number of Android emulators (hence, users) that can be supported in one EC2 instance [22]. Figure 19 shows the configuration of an AVD. Except for their names, all the AVD use the same configuration. After all the above configurations are complete, we stop this instance and create an Amazon Machine Image (AMI) [7] from it. All the frontend instances used in conducting the large-scale testing will be launched from this AMI.

5.2.3 Test Conducting

In this section, we introduce the steps to conduct ACT.

5.2.3.1 Launch and terminate a specified number of frontend instances from that AMI

We have written a standalone Java program that uses AWS Java API. We put Access Key ID and Secret Access Key associated with our AWS account as the credentials in this program to access AWS. This program is executed on a PC and is used for launching a specified number of frontend EC2 instances and then terminating them later. When launching instances, AWS Java API allows us to specify many parameters, such as the number of instances, which AMI to use, which security group they belong to, etc. We launch frontend instances one by one in a manner that there is a fixed time gap between their launches. We also terminate those instances in the



AVD Name:	oadTesting01
Device:	5.1" WVGA (480 × 800: mdpi)
Target:	Google APIs (Google Inc.) - API Level 10
CPU/ABI:	ARM (armeabi)
Keyboard:	✓ Hardware keyboard present
Skin:	✓ Display a skin with hardware controls
Front Camera:	None
Back Camera:	None
Memory Options:	RAM: 512 VM Heap: 24
Internal Storage:	200 MiB
SD Card:	● Size: 500 MiB ○ File:
Emulation Options:	✓ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

Figure 19: The configuration of an AVD

same order and manner.

5.2.3.2 *Launch Android emulators on each frontend instance*

Because all the frontend instances used in the testing are launched from the same AMI, they will be automatically logon once their operating systems are ready. Then the downloader file is automatically executed, downloading all the other files and executing the launcher file. Then the launcher file launches a specified number of Android emulators one by one from Windows terminal using tools provided by Android SDK [12]. The launcher file will then wait for some time for those Android emulators to be ready, and then launch monkeyrunner instances one by one, each with a Python script file.

*5.2.3.3 Launch *Mobile app on each Android emulator*

A Python script file is used to control what monkeyrunner will do to applications on an Android emulator. There are three kinds of user events that monkeyrunner can generate on Android applications: a drag event, a keyboard press event and a finger touch event. Monkeyrunner is also able to launch a specified application on android emulator and take screenshot. We use monkeyrunner along with the Python script file to unlock the keypad of the Android emulator and then launch *Mobile application.

*5.2.3.4 *Mobile login and mobile application login*

Some of the mobile applications might require login before users can use them. In the case of this testing, we need to log into *Mobile and the application mobilized by it. All logins have some similar characteristics: there are some input fields that once they are touched, users can type in username and password; some applications may also require the user to choose some item from a drop-down list; finally, a confirmation button is touched and login process finishes after some time. In this testing, we use Pixel Perfect [29] to determine the exact coordinates of input fields in order to write code in Python files to touch them, and then send username and password hard-coded in Python file to these input fields on the Android application through monkeyrunner. Choosing whatever needed to login and touching the confirmation button can be done by sending touch events at their coordinates on the screen.

5.2.3.5 Randomized testing

We conduct randomized testing after we complete the mobile application login. By randomized testing, we mean we generate a specified number of random user events and execute them on the Android application through monkeyrunner. Monkeyrunner by itself does not provide the capacity to generate and execute random user events; it can only be used to generate and execute deterministic user events though its Python API. However, we can use our own algorithm to generate pseudo-random user events and then execute them on the mobile application. The algorithm we use to generate and execute a pseudo-random user event in monkeyrunner is: Algorithm 1: The generation and execution of a pseudo-random user event

Generate a random number n in the range of [1, 6]:

- If n is less than 3, this is a drag event
  - Generate another four random numbers x1, y1, x2, y2, where x1, x2 are small than the screen width and y1, y2 are smaller than the screen height
  - Generate and execute a drag event from (x1, y1) to (x2, y2)
- Otherwise if n is 3, this is a keyboard press event
  - Generate another random number k to determine which key to press in this event
  - Generate and execute a keyboard press event with this key
- Otherwise this is a finger touch event
  - Generate another two random numbers t1 and t2, where t1 is small than the screen width and t2 is smaller than the screen height
  - Generate and execute a touch event at (t1, t2)

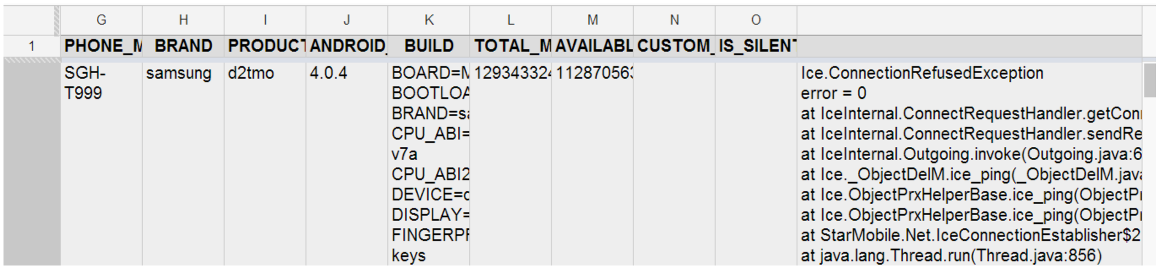
The above process can be repeated many times to generate and execute sufficient amount of random user events on the mobile application being tested.

5.2.4 Additional Features

In this section, we introduce two additional features we've built into the testing. They can be used in conducting functional testing, performance testing and stress testing in a large-scale manner.

5.2.4.1 *Exceptions reporting*

We want to be notified whenever an exception occurs in the mobile application being tested. We use ACRA [2] for the *Mobile frontend. ACRA is an open-source library enabling Android Application to automatically post their crash reports to a Google Docs form. It is targeted to android applications developers to help them get data from their applications when they crash or behave erroneously. For the *Mobile backend server, we send an email to a support email account whenever any UI and non UI-related exception occurs. Figure 20 shows an exception reported by ACRA to a Google Spreadsheet.



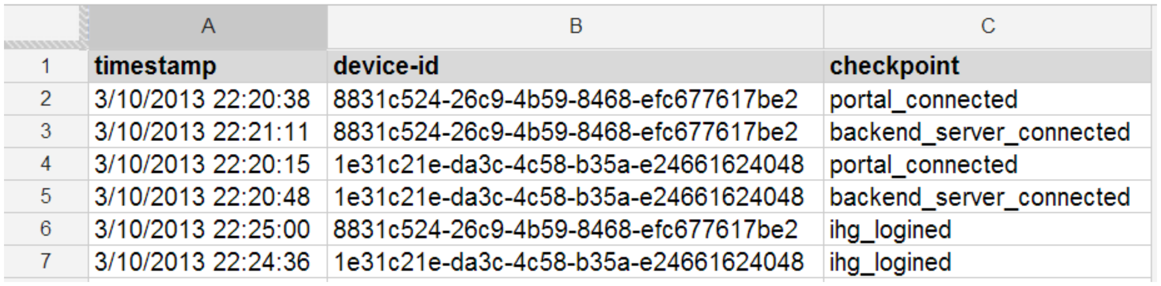
	G	H	I	J	K	L	M	N	O	
1	PHONE_M	BRAND	PRODUCT	ANDROID_	BUILD	TOTAL_M	AVAILABL	CUSTOM_	IS_SILENT	
	SGH-T999	samsung	d2tmo	4.0.4	BOARD=M BOOTLOA BRAND=sa CPU_ABI= v7a CPU_ABI2 DEVICE=d DISPLAY= FINGERPR keys	12934332	11287056			Ice.ConnectionRefusedException error = 0 at IceInternal.ConnectRequestHandler.getCon at IceInternal.ConnectRequestHandler.sendRe at IceInternal.Outgoing.invoke(Outgoing.java:6 at Ice._ObjectDelM.ice_ping(_ObjectDelM.java at Ice.ObjectPrxHelperBase.ice_ping(ObjectPr at Ice.ObjectPrxHelperBase.ice_ping(ObjectPr at StarMobile.Net.IceConnectionEstablisher$2 at java.lang.Thread.run(Thread.java:856)

Figure 20: An exception reported by ACRA

5.2.4.2 *Checkpoints reporting*

We want to be notified when the execution of a mobile application reaches some checkpoints. Hence, we implement a checkpoint reporting mechanism. Three types of checkpoints are reported: *Mobile user portal connected, *Mobile backend server connected and *Mobile-mobilized application login. We implement these at the *Mobile frontend by incorporating some Android code which uses google-spreadsheet-lib-android [18] into the *Mobile frontend codebase. Google-spreadsheet-lib-android is a third party Java open-source library, which helps to develop Android applications which access Google Spreadsheets. It uses Google Documents API [17] and Google Spreadsheets API [19] to allow developers to create, view, edit, and delete Google

Spreadsheets associated with the user accounts on mobile devices. The reporting code we incorporate into the *Mobile frontend codebase will be executed whenever a checkpoint is hit. It will then insert a new record entry into a worksheet in a Google spreadsheet. Figure 21 shows some checkpoint reporting results in Google Spreadsheet.



	A	B	C
1	**timestamp**	**device-id**	**checkpoint**
2	3/10/2013 22:20:38	8831c524-26c9-4b59-8468-efc677617be2	portal_connected
3	3/10/2013 22:21:11	8831c524-26c9-4b59-8468-efc677617be2	backend_server_connected
4	3/10/2013 22:20:15	1e31c21e-da3c-4c58-b35a-e24661624048	portal_connected
5	3/10/2013 22:20:48	1e31c21e-da3c-4c58-b35a-e24661624048	backend_server_connected
6	3/10/2013 22:25:00	8831c524-26c9-4b59-8468-efc677617be2	ihg_logined
7	3/10/2013 22:24:36	1e31c21e-da3c-4c58-b35a-e24661624048	ihg_logined

Figure 21: Checkpoint Reporting Results in Google Spreadsheet

CHAPTER VI

PERFORMANCE EVALUATION OF ACT

We use four criteria to evaluate ACT: correctness, scalability, cost-efficiency and time-efficiency.

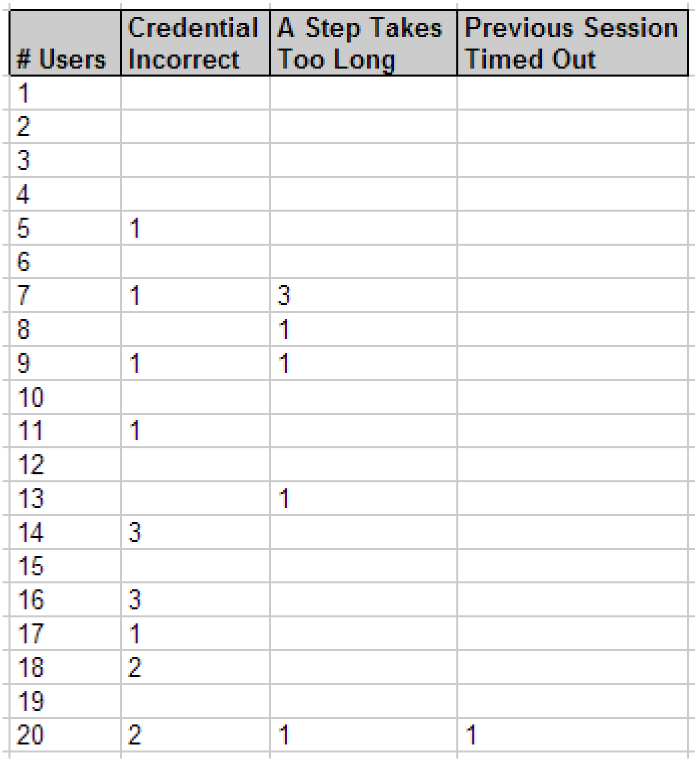
6.1 Correctness

Whether we can get useful evaluation information about the mobile application being tested is determined by the correctness of the testing solution itself. By a correct testing solution, we mean that it should be able to correctly tell whether each user of a particular mobile application passes or fails the test. If a user fails, it should be able to provide sufficient information for testers to diagnose the cause of the failure as well.

Monkeyrunner script file allows us to capture the screenshots of the Android emulator it controls. We configure the monkeyrunner script file in the frontend instance template so that all the frontend instances will save screenshots to the same Dropbox folder. Thus we have a central place to view all the screenshots for all the users supported in any ACT round. Each screenshot is captured whenever a particular step in a ACT round is complete in each Android emulator. The name of each screenshot file is prefixed with a random number so that we can distinguish between the screenshots for each Android emulator. Each Android emulator would generate 10 screenshots so that we know when a particular round of ACT is complete by counting the number of screenshots in that Dropbox folder.

By examining the screenshots, we know whether a particular ACT round passes or not. If it passes, both the ACT and the mobile application being tested are correct. If it fails, these screenshots can also tell us whether this failure comes from the ACT itself or the mobile application.

We conduct 20 rounds of ACT with 1 to 20 users to evaluate its correctness. Figure 22 shows the experiment results. We can see from the figure that not all ACT rounds pass because of three kinds of failures. All these failures come from *Mobile, which is being tested in these 20 ACT rounds. All the ACT rounds are able to tell whether each user passes the test or not, and when any user fails the test, they are able to tell what are the causes of their failures. Therefore, the correctness of ACT is 100%, while the correctness of *Mobile is 89.05% (187 out of 210 users pass the test).



# Users	Credential Incorrect	A Step Takes Too Long	Previous Session Timed Out
1			
2			
3			
4			
5	1		
6			
7	1	3	
8		1	
9	1	1	
10			
11	1		
12			
13		1	
14	3		
15			
16	3		
17	1		
18	2		
19			
20	2	1	1

Figure 22: ACT correctness evaluation experiment results

6.2 *Scalability*

The goal of ACT is to enable testers to conduct large-scale testing of the concerned mobile application at the UI level, hence the number of users ACT can test the mobile application against, i.e., the scalability of ACT, is naturally one of the evaluation criteria.

We conduct 20 rounds of ACT with 1 to 20 users to evaluate its scalability. We record the start time of each ACT round, which is defined as the time when we run the standalone Java program to launch a specified number of frontend instances. We also record the end time of each ACT round, which is defined as the time when all the users in this round finish the testing. We can know when all the users finish the testing by counting the number of screenshots in the Dropbox folder. We then calculate the duration of each ACT round from its start time and end time.

Figure 23 shows the durations of these 20 ACT rounds. We can see from the figure that the duration for each ACT round does not increase with the increase of the number of users. Therefore, ACT is scalable in terms of the time it needs. However, the cost of each ACT round increases with the increase of the number of users. This may result in a huge cost that would limit the number of users ACT can support. Therefore, we discuss and evaluate its cost-efficiency in the next section.

6.3 Cost-efficiency

Using Amazon EC2 services is the only cost in ACT. There are several types of cost in using Amazon EC2 services (such as using EC2 instances, using volume, data transfer, etc), but we will only focus on reducing the cost of using EC2 instances since other types of cost are neglectable. We want to have the flexibility of launching a specified number of a specific type of EC2 instances whenever we want, so we choose to use Amazon EC2 On-Demand Instances [5] in ACT. There are 18 types of EC2 instances [4] that can be used as On-Demand Instances, each with different capacity (CPU, memory, etc) and different cost per hour. On most of the 18 types of instances, we can launch more than one Android emulators (hence support more than one users). The duration of each round of ACT is not affected by the type of EC2 instance used.

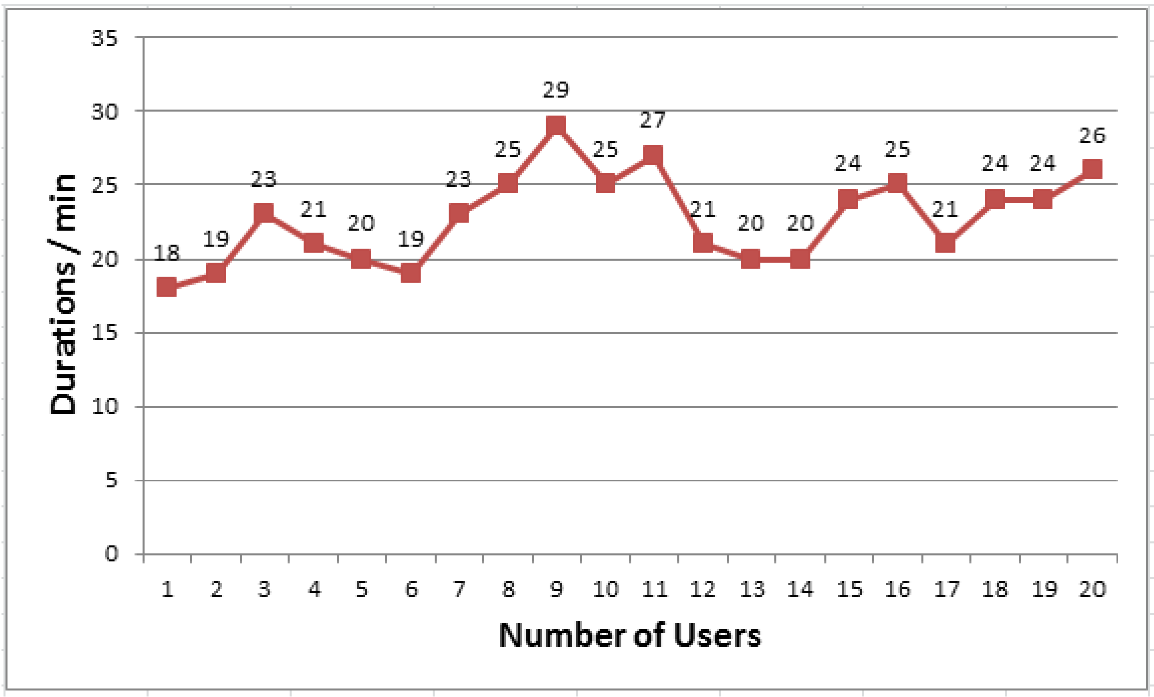


Figure 23: Duration of each ACT round

Therefore, in order for ACT to be cost-efficient, we need to determine how many users each type of EC2 instances can support without compromising the correctness of ACT.

Figure 24 shows cost per user hour for each Amazon EC2 instance type. Each instance type is represented with its API name in the figure. We can see that the most cost-efficient instance type has API name ”t1.micro”, which represents micro instances. Micro instances only costs $0.02 for supporting one user in one hour and can only support one user. Because micro instance is the most cost-efficient instance type, we always use this type of instance in conducting and evaluating ACT with other criteria.

6.4 Time-efficiency

We should also care about how time-consuming our ACT is. In terms of the speed at which ACT is conducted, we support two kinds of testings, a normal ACT and an accelerated ACT. In a normal ACT, the test will be conducted as fast as the users’

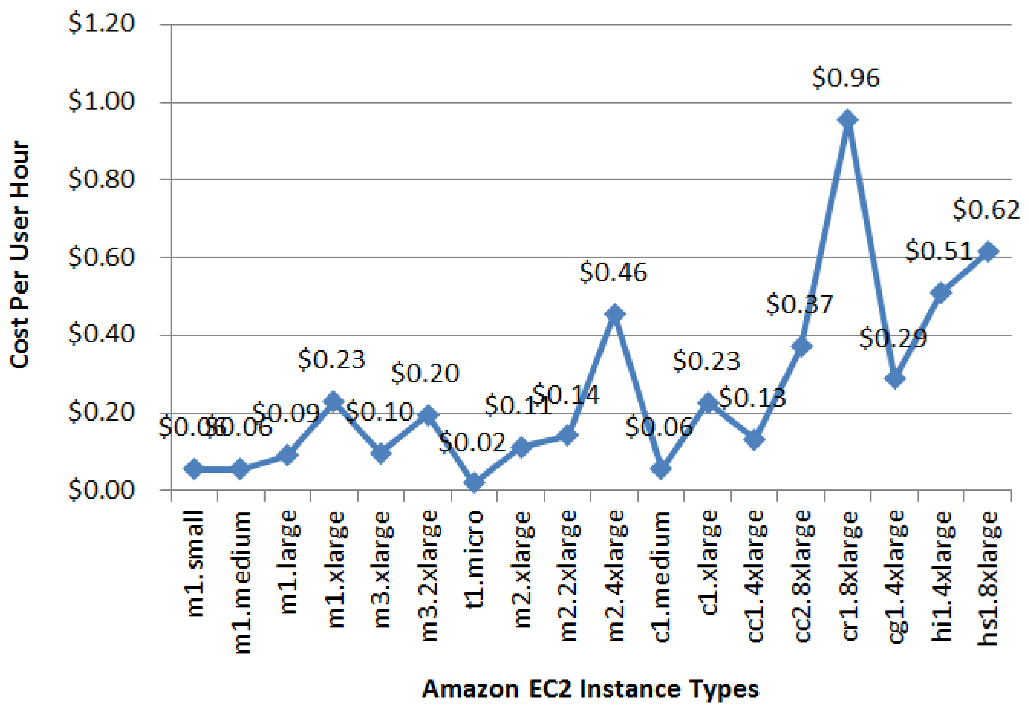


Figure 24: Cost per user hour for each Amazon EC2 instance type

interactions with the mobile application. Normal ACT tries to mimic how users use the concerned mobile application as closely as possible. But sometimes testers may only be concerned about some checkpoint or end results of the mobile application so that they would like to conduct the test as soon as possible. Because of that requirement, we also present accelerated ACT and use time-efficiency (how fast an accelerated ACT can be conducted) as one of the evaluation criteria.

We base our accelerated ACT implementation on *Mobile. Once *Mobile is launched, users will first see an activation dialog. Users will need to type in their *Mobile activation credentials to activate *Mobile. This is just a one-time process but we still include it in the ACT acceleration experiment. When this activation is done, the *Mobile frontend will try to connect to a *Mobile backend. Once this connection is done, users will see a container page. This page contains icons that represent all the applications mobilized by *Mobile for that user. Users can then click on an icon to launch the corresponding mobile application. Once the mobile application is launched, users would usually see a login page for that mobile application. Users can then type in their credentials for that mobile application to login. When users log into the mobile application, we consider accelerated ACT is complete in our implementation.

We divide the whole ACT procedure into ten steps: launching frontend instances, executing test scripts, launching Android emulators, connecting to Android emulators, launching *Mobile, *Mobile activation, connecting to *Mobile backend, launching mobile application, entering login credentials and mobile application login. Of the above ten steps, four of them are under our control and can be accelerated: launching *Mobile, connecting to *Mobile backend, launching mobile application and mobile application login.

Now we introduce the implementation of accelerated ACT. In a high level, since we can get checkpoint information in the *Mobile frontend Android code but the progress of ACT is controlled by the monkeyrunner Python script, the acceleration requires the communications between them. We have set up four checkpoints in *Mobile frontend Android code based on the occurrences of some UI elements. Each of the four checkpoints is used to notify the monkeyrunner Python script that one of the four acceleratable steps is complete, so that the next step can start immediately. Whenever a checkpoint is hit, *Mobile frontend will create a checkpoint file in the SD Card folder of the Android emulator. The monkeyrunner Python script keeps checking whether the checkpoint file exists or not by invoking "adb pull" command [9] periodically. "adb pull" command is used to copy files from the local file system of a device or emulator to the local file system of the underlying development PC. It has return code to indicate whether this can be done or not based on the existence of the checkpoint file. Therefore, once a checkpoint is hit, the monkeyrunner Python script will know from invoking "adb pull" command and proceed to conduct the next step of accelerated ACT.

In order to evaluate the time-efficiency of accelerated ACT, we have designed three scenarios: brute-force, manual and accelerated. In a manual scenario, instead of implementing acceleration through the above approach, we conduct many rounds of ACT and get a sense of how long each of these four steps would take. Then we hard-code how long the monkeyrunner Python script should wait for each of the four steps to finish and then execute the next step. In a brute-force scenario, we don't have any information about how long each step would take and just use the longest time determined by the manual scenario for all four steps. In an accelerated scenario, we first implement acceleration through the above approach and conduct a round of ACT with ten users. We then record and calculate the average time needed by each step in this round of ACT.

Figure 25 shows the comparison of the above three scenarios. Compared to brute-force scenario, we can see from the figure that our acceleration implementation can significantly reduce the time needed by ACT. Moreover, it is even more time-efficiency than ACT in manual scenario.

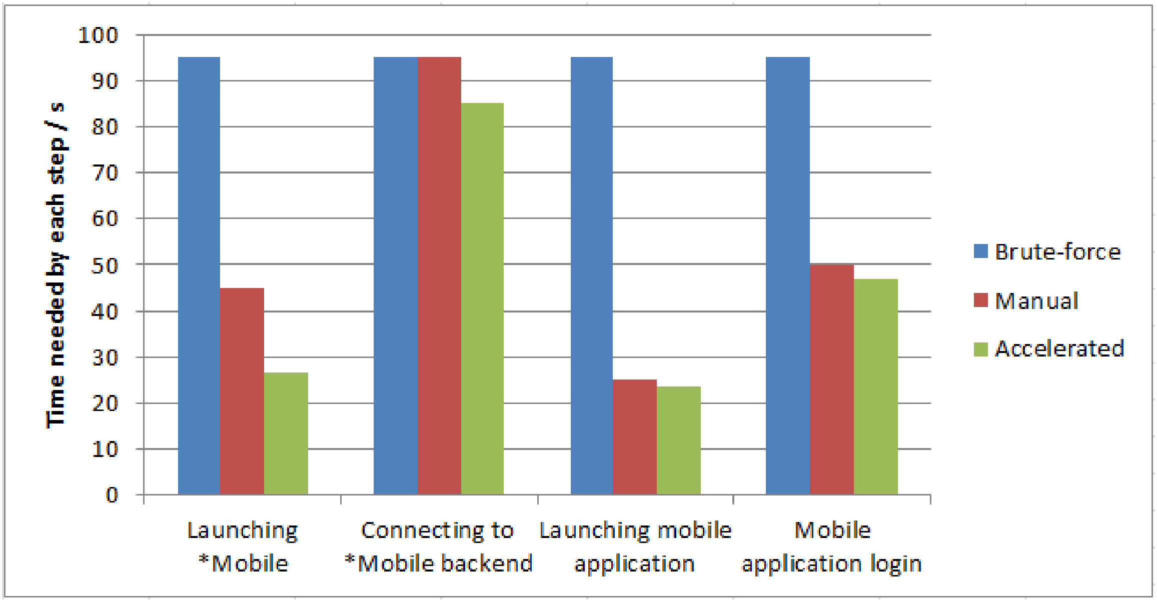


Figure 25: ACT Acceleration Experiment Results

CHAPTER VII

CONCLUSIONS

In this thesis, we focused on two problems in mobile application development: contextualization and large-scale testing. We identified the limitations of current contextualization and testing solutions. Advanced-remote-computing-based mobilization does not provide context awareness to the mobile applications it mobilized, so we presented contextify to overcome this limitation. Evaluation results showed that contextify-contextualized applications require less time and effort for users to complete tasks. Current mobile testing solutions cannot conduct tests at the UI level and in a large-scale manner simultaneously, so we presented automated cloud computing (ACT) to achieve this goal. Evaluation results showed that ACT can support a large number of users and it is stable, cost-efficiency as well as time-efficiency.

7.1 Main Contributions

- We presented a solution called contextify to provide context awareness to mobile applications mobilized by advanced remote computing solutions. Contextify can save users' time and burden in using mobile applications without rewriting the mobile applications.
- Based on contextify, we designed five use cases and implemented two of them as an overlay on an Android application frequently used by Georgia Tech students, faculty and staff.
- We evaluated and demonstrated the effectiveness of contextify by comparing users' time and effort in using normal application and contextified application under two different scenarios. We also conducted users survey and showed that

user experience has been improved.

- We presented a solution called automated cloud computing (ACT) to allow testers to conduct mobile application testing at the UI level and in a large-scale manner at the same time.

- We implemented ACT by launching Android emulators in compute instances in the cloud and use monkeyrunner for automation. We also implemented additional features such as exceptions reporting and checkpoints reporting to aid the conducting of functional testing and performance testing.

- We evaluated and demonstrated the effectiveness of ACT against four criteria: correctness, scalability, cost-efficiency and time-efficiency.

REFERENCES

[1] "About us page for utest." http://www.utest.com/about.

[2] "Acra." http://code.google.com/p/acra/.

[3] "Amazon ec2 console dashboard." https://console.aws.amazon.com/ec2/home.

[4] "Amazon ec2 instance types." http://aws.amazon.com/ec2/instance-types/.

[5] "Amazon ec2 pricing." http://aws.amazon.com/ec2/pricing/#on-demand.

[6] "Amazon elastic compute cloud (amazon ec2)." http://aws.amazon.com/ec2/.

[7] "Amazon machine images (amis)." https://aws.amazon.com/amis.

[8] "Android bluetooth apis." http://developer.android.com/guide/topics/ connectivity/bluetooth.html.

[9] "Android debug bridge." http://developer.android.com/tools/help/adb.html.

[10] "Android emulator." http://developer.android.com/tools/help/emulator.html.

[11] "Android location and maps guide." http://developer.android.com/guide/topics/location/index.html.

[12] "Android sdk." http://developer.android.com/sdk/index.html.

[13] "Apple ahead of google with 25b app downloads." http://www.informationweek.com/mobility/smart-phones/apple-ahead-of-google-with-25b-app-downl/232601991.

[14] "Cloudtest with the global test cloud for load testing." http://www.soasta.com/products/campaign-global-test-cloud/.

[15] "The comscore 2010 mobile year in review." http://www.comscore.com/Insights/Presentations_and_Whitepapers/2011/2010_Mobile_Year_in_Review.

[16] "Global mobile statistics 2013." http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#uniquesubscribers/.

[17] "Google documents list api version 3.0." https://developers.google.com/google-apps/documents-list/.

[18] "google-spreadsheet-lib-android." http://code.google.com/p/google-spreadsheet-lib-android/.

[19] "Google spreadsheets api version 3.0." https://developers.google.com/google-apps/spreadsheets/.

[20] "How to turn on automatic logon in windows." http://support.microsoft.com/kb/324737.

[21] "Idc: Q4 2010 smartphone shipments increase 87.2%." http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22689111#.UWggCaV0Uy0.

[22] "it seems too many emulator instances are running on this machine. aborting." https://groups.google.com/forum/?fromgroups=#!topic/android-beginners/oD_g3i7EFzg.

[23] "Java developer center for amazon web services." http://aws.amazon.com/java/.

[24] "Load testing mobile applications in neoload." http://www.neotys.com/product/mobile-load-testing.html.

[25] "Managing virtual devices." http://developer.android.com/tools/devices/index.html.

[26] "Mobile application testing in contus." http://mobileappstesting.contussupport.com/.

[27] "monkeyrunner." http://developer.android.com/tools/help/monkeyrunner_concepts.html.

[28] "Normal or average walking, jogging and swimming speed of a human." http://www.yogawiz.com/blog/walking/normal-walking-speed.html.

[29] "Optimizing your ui." http://developer.android.com/tools/debugging/debugging-ui.html.

[30] "Simulate realistic user load in agileload." http://www.agileload.com/agileload-features/product-presentation/simulate-realistic-user-load.

[31] "Smartphones have conquered pcs." http://money.cnn.com/2011/02/09/technology/smartphones_eclipse_pcs/index.htm/.

[32] "Smartphones to overtake feature phones in u.s. by 2011." http://blog.nielsen.com/nielsenwire/consumer/smartphones-to-overtake-feature-phones-in-u-s-by-2011/.

[33] "T-square project site." http://info.t-square.gatech.edu/.

[34] "Vnc." http://www.realvnc.com/.

[35] Anind K. Dey, G. D. A. and Salber, D., "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interactions (HCI) Journal*, 2001.

[36] DEY, A. K., "Understanding and using context," *Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing 5*, 2001.

[37] Dey, A. K. and Abowd, G. D., "Towards a better understanding of context and context awareness," in *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, p. 304307, London, UK. Springer-Verlag, 1999.

[38] Flanagan, J. A., *Context Awareness in a Mobile Device: Ontologies versus Unsupervised/Supervised Learning.* Nokia Research Center, 2005.

[39] Likert, R., "A technique for the measurement of attitudes.," in *Archives of Psychology*, New York : [s.n.], 1932.

[40] Lum, W. Y. and Lau, F. C., "A context-aware decision engine for content adaptation," *IEEE Pervasive Computing*, 2002.

[41] Mantyjarvi, J., Huuskonen, P., and Himberg, J., "Collaborative context determination to support mobile terminal applications," *IEEE Pervasive Computing*, 2002.

[42] Ogata, H. and Yano, Y., "Context-aware support for computer-supported ubiquitous learning," in *IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE)*, 2004.

[43] Ranganathan, A., Campbell, R. H., Ravi, A., and Mahajan, A., "Conchat: A context-aware chat program," *IEEE Pervasive Computing*, 2002.

[44] Tsaoc, C.-L., *Rapid Application Mobilization and Delivery for Smartphones.* PhD thesis, Georgia Institute of Technology, 2012.

[45] Uta Christoph, Janno von Stlpnagel, K.-H. K. and Terwelp, C., "Context detection on mobile devices," in *Context-Systems Design, Evaluation and Optimisation (CoSDEO)*, 2011.

[46] Voida, S., Mynatt, E. D., MacIntyre, B., and Corso, G. M., "Integrating virtual and physical context to support knowledge workers," *IEEE Pervasive Computing*, 2002.