An Approach to Mobile Application Testing Based on Natural Language Scripting

Chuanqi Tao[①], Jerry Gao[②③], Tiexin Wang[①]

① College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

②Department of Computer Engineering , San Jose State University, USA

③Taiyuan University of Technology, Taiyuan, China

Correspondence to: jerry.gao@sjsu.edu

Abstract: With the rapid advance of mobile computing technology and wireless networking, there is a significant increase of mobile subscriptions. This brings new business requirements and demands in mobile software testing, and causes new issues and challenges in mobile testing and automation. As there are multiple platforms for diverse devices, engineers suffer from the different scripting languages to write platform-specific test scripts. In addition, a unified automation infrastructure is not offered with the existing test platform. This paper proposes a novel approach to mobile application testing based on natural language scripting. A Java-based test script generation approach is developed to support executable test script generation based on the given natural language-based mobile app test operation scripts. A prototype tool is implemented based on some open sources. Finally, the paper reports empirical studies to indicate the feasibility and effectiveness of the proposed approach.

KEYWORDS: mobile application testing; test automation; scenario-based testing; behavior-based testing

1. Introduction

Today mobile computing is the beginning of another revolution that is going to sweep the consumer industry with its fast paced growth. The demand for smart phones and tablets coupled with their expensive data plans is already soaring high and is a trillion dollar industry. As mobile APP and mobile application vendors, they have encountered the following critical issues. Testing mobile apps on different mobile platforms and various devices becomes very costly and tedious due to the fast upgrading of mobile devices, and rapid updates of mobile platforms and technologies. Testing mobile web applications on diverse mobile browsers on different devices become very difficult and costly due to increasing mobile scalability, constantly updates of mobile devices and mobile technologies, diversity of involved mobile web technology, and fast upgrading mobile application services [1, 2, 3, 18].

Most of the present research work focuses on providing solutions to the technical problems in the following areas: a) mobile app white-box testing methods; b) GUI-based test technique for mobile apps [5]; c) usability testing [6]; d) ad hoc scripting test tools; e) wireless network testing [9]. There are two challenges in mobile testing: the first issue is too costly to deal with diversity of mobile test environments on varieties of mobile devices, and the other is the lack of cost-effective method and platforms to support a unified mobile test automation crossing different mobile platforms on diverse devices. Most of the existing mobile testing tools support GUI-based functional testing and few support load/performance testing. Currently, a unified automation infrastructure is not offered with the existing test tools. In addition, there is lack of well-defined mobile test scripting method to deal with the massive multiple mobile test running. Therefore, test automation central control is needed to support behavior-based testing or scenario-based testing at multiple levels.

In this paper we develop practical solutions and design a unified mobile test platform to address the needs and limitations. The intent is to develop a unified system that enables test engineers to dynamically test the mobile applications without the dependency on any scripting language. The proposed approach will enable test engineers to define the scenarios to be tested in plain natural language that supports seamless and efficient testing of mobile applications. In this approach, we attempt to design a system capable of testing the properties of the application automatically once the scenarios are written for a set of features from natural language. As the test cases can be purely user written stories, we can simple use natural language to write the functionality and scenarios to test, and then script the code to perform those tests automatically. Further, the code to execute these tests would be a part of step-definitions in the overall test automation framework.

The paper is structured as follows. The next section presents the background and related work. The architecture of the BDD-based natural language processing analysis is discussed and the designed and implemented prototype tool for the proposed approach is presented in Section 3. Case studies of testing on several mobile apps are shown in Section 4. Conclusion and future work are summarized in Section 5.

2. Background and Related Work

2.1 Background

Behavior driven development (BDD) includes a work flow which requires the software developers to test the behavior

DOI reference number: 10.18293/SEKE2017-170

of a piece of code as and when it is developed [21, 23]. It is used by the stakeholders to verify if the product approach is correct or wrong. BDD is based on scenarios to check the validity of the system. Using this design flow based on the scenarios will allow us to analyze the whole piece of code, scenario by scenario. This not only helps developers understand the functioning of the component, the stakeholders can also verify the step process. BDD is written in natural language which is easier for both the parties to understand and verify the functionality. Natural language allows the developers to see if there is any ambiguity in the thought process of a system due to a complicated process. BDD works in a sentence by sentence approach rather than taking the whole code as a whole. It allows everyone to see which line is getting the correct result and which is not. As the test cases can be purely user written stories, there is no need to write test cases in code. Therefore, we can use natural language to write the functionalities and scenarios to test, and then script the code to perform those tests automatically. Thus, we can easily write human understandable user stories and this can enable even beginners to test the applications. Further, the code to execute these tests would be a part of step-definitions. This acts as a layer of abstraction and promotes a series of advantages and efficiencies in mobile application testing.

2.2 Related Work

Up to today, many papers have been published to address different testing areas in mobile applications. These areas include such as white-box and black-box testing, quality-of -service testing, wireless connectivity testing, and test automation frameworks.

Existing white-box testing methods are still applicable to mobile apps. For example, Java Pathfinder [1] is a mobile program verification tool supporting white-box mobile Java program testing. Engineers can use this tool to detect race conditions and deadlocks based on UML state machines and symbolic execution. Mahmood and his colleagues [2] use a white-box approach to generate test cases with two program-based models (call graph and architectural) to achieve mobile program code coverage. Many black-box testing techniques are useful in mobile apptesting. Random testing and the scenario-based testing method [3] are good examples. GUI-based testing has been discussed in numerous papers. For instance, Anand and his colleagues [4] introduced an automated testing approach to validating mobile GUI event sequences for smart-phone apps. Similarly, Amalfitano and his colleagues [5] presented a tool AndroidRipper that uses an automated GUI-based technique to test Android apps in a structured manner.

There are many tools that support BDD such as cucumber, jBehave, twist etc. In recent times, a large number of mobile testing tools have been developed to support mobile development [17, 21, 23]. As the number of mobile products is increasing, the testing of these products is important. We need to choose the testing tool that best suits the product. Thereby, increasing demand for mobile devices has lead to the needs for developing tools to test them at a high level. Based on our research, we found that most of the testing tools either perform the GUI testing or load and performance testing. But all the tools have some limitations. For instance, many tools require engineers to learn different scripting languages in order to write platform specific and tool specific scripts which would be difficult for them. However, engineer still need new test models to address special needs in testing mobile applications.

Unlike the existing research, our goal in this paper is to address the growing needs of the mobile market like the use of natural language to test the mobile applications. Our approach aims in providing solution to such problems and our research has a wide impact on the market which keeps on changing rapidly.

3. BDD-Based Natural Language Scripting for Mobile Application Test Generation

The order of behavior driven development in the form of acceptance tests is as follows.

(a) A scenario is described to define the action or behavior of the system to be tested;
(b) Step definition is the part where the natural language is converted into the actual working code based on the mapping of the constructs of the natural language. A specific regular expression is used to determine the code which is to be executed on reading the sentence;
(c) A code skeleton is needed to comply with the tool that is being used, so that efficient codes can be written and run;
(d) Codes in the code skeleton are then run which will show us the results. This will verify if the code has passed or failed depending on the scenario.

We used the tool *cucumber* (*https://cucumber.io/to*) to run the nature language scripts written in *Gherkin* language. The server side code i.e., the step definitions for *cucumber* are written in Ruby language which contains all the definitions and classes needed to test the functionalities of the mobile app. Cucumber runs and points out the steps which have been used but not yet defined and prompts the user to define each of those steps. These step definitions can be written in any language where we used Ruby for defining steps in our approach. The framework Calabash (*http://calabash.sh*) is used to support automated tests written in ruby and Gherkin. The framework contains all the scenarios in the feature file written in nature language, Gherkin. It also contains the step definitions which act as a server side code written in Ruby language. A number of self-defined functions are used in our current approach.

In brief, the process of BDD for mobile testing here includes scenario description, step definition, code skeleton, and implementation. Figure 1(a), (b), (c), and (d) shows the

DOI reference number: 10.18293/SEKE2017-170

examples of BDD in the scenario of *telephone call*.

Scenario: *Placing a call*

* Ada picks up the receiver from the telephone
* She dials the number 6-345-789
* The telephone places a call

(a) Scenario

```
Given /^Ada picks up the receiver from the telephone$/ do
  @telephone = Telephone.new
  @receiver = @telephone.pickUp
end
```

(b) Step Definition

```
class Telephone
  def pickUp
  end
end

class Receiver
end
```

(c) Code Skeleton

```
class Telephone
  attr_reader :receiver

  def initialize
    @receiver = Receiver.new
  end

  def pickUp
    @receiver
  end
end
```

(d) Implementation

Figure 1 A sample BDD process

3.1 Modeling Semantic Relation of Natural Language

In order to obtain a semantic relation between the keywords of the defined language, we need to put together the sentence in a syntactical manner. Both these parsers will help us create a Phrase Structure tree (PST). This phrase structure tree will be the backbone of our natural language. Our approach will try and break all the keywords to be used into skeleton codes. We firstly divide the language into grammatical components, and then define each and every grammatical component. Secondly, a class and function is tied to every component. Thirdly, we define the code for each and every specific tie up. Finally, we run the scripts to return the behavior. Figure 2 shows some examples on how to break a sentence into a PST. Figure 3 presents some key words in our language (bold words). When a part of natural language has been written, a parser is required to break the language into components and relate with the skeleton code in the background.

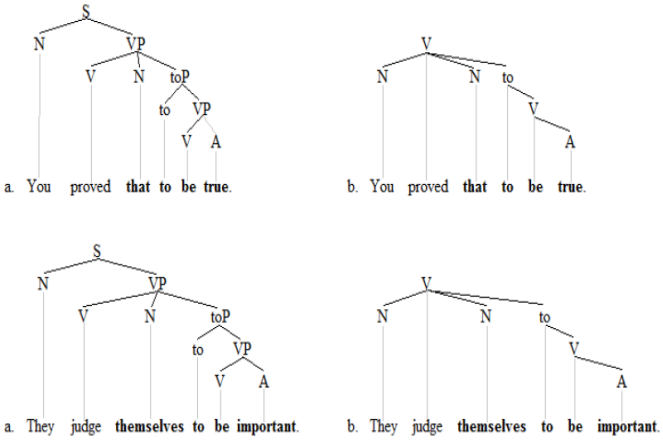


Figure 2 PST examples

```
Feature: Multiple Scenario
In order to extend cucumber functionality
as a project developer
I want to include multiple test scenarios

Background: Given are a set of scenarios in a hierarchical order

@user @app @scenario @environment

Scenario Outline: Check network connectivity
And check another scenario for security certificates
And <another scenario after that>

Given security certificates are exchanged
And <another scenario is completed>
And network connectivity checks out

When connection is made by the user
And connection is fully authenticated

Then run the Selenium tests
And Appium tests
And Return the results in the console
```

Figure 3 A sample keywords in our language

3.2 Design and Implementation for the Approach

This section presents the developed software system architecture and explains about the involved components and their relations and connectivity.

Figure 4 shows the workflow of the designed system. The blue colored components are developed in our approach. Most of the yellow colored components can be built from the existing work and some open source tools. The Gherkin extension is implemented where the majority of the functionality and logic is defined. The intelligence goes in extending the behavior of Cucumber by defining extensions based on scenarios written in nature language. The part enclosed in the dotted lines is the work flow of test scripting in the solution. Our approach aims to design a prototype systematic tool which can support to generate mobile application testing scripts based on the requirements of the user input. The user input determines which script to be selected from the database and which environment to set up. We analyze the user inputs and categorize them into three different parts. Then these parts will be segregated into the Gherkin extension which forms the scripts based on the user inputs. Next, some implemented key components and used technology are discussed in details.

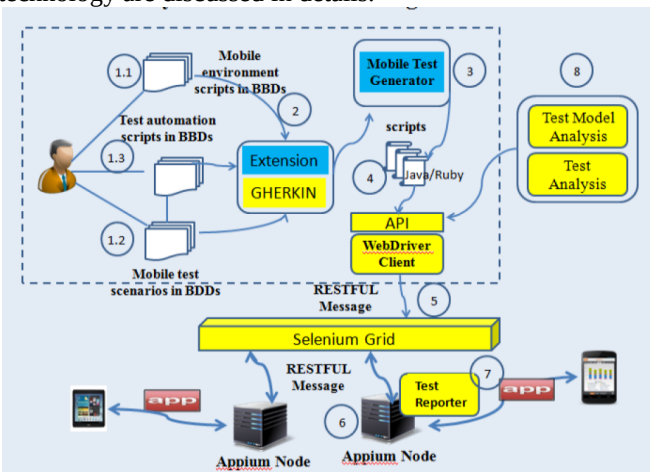


Figure 4 The workflow of the designed system for mobile application testing

DOI reference number: 10.18293/SEKE2017-170

User inputs - Three types of user inputs are shown in Figure 4. The first set of inputs refers to the environment parameters. These set of options is predefined by the system user and would be allowed to choose the options according to the test suite or the test script. Test automation scripts in behavior driven development will be supplied by the user for automating test scripts execution. These scripts are being handled by some other team. The third input would be the required test scenarios for testing the application. The test scenarios are implemented at a multiple level.The existing work only focuses on one level scenario that was being executed. In this approach, we try and implement the scenarios at multiple levels.

Gherkin extension - Gherkin is a simple natural language based programming language. It does not have a very complex and detailed syntax. Only a few keywords are required to use gherkin as a language. The input supplied by the user is processed by the Gherkin extension. When we run the gherkin scripts in cucumber, it generates a report based on the keywords and sentences provided no matter the script will run or not. If it runs, then we check if it behaves in the way we have defined in gherkin script. After that, the related information is sent to the Mobile Test generator (the component shown in Figure 4) for execution.

Mobile test generator - In this mobile test generator, all the additional functionality and logic is applied to define the test scripts. The Cucumber understands the natural language based scripts, while internally uses Java/ Ruby language for procession. The interactions among the servers happen by using the defined API's.

Defining Steps in Cucumber- We used the tool cucumber to run our natural language scripts written in Gherkin language. The server side code i.e., the step definitions for Cucumber are written in Ruby language which contains all the definitions and classes needed to test the functionalities of the mobile app. The sample features and scenarios are illustrated as follows.

Cucumber Components- Features and Scenarios - Following this approach for defining the steps will make the steps/scenarios more complete and readable though the order is not critical. Cucumber runs and points out the steps which have been used but not yet defined and prompts the user to define each of those steps. These step definitions can be written in any language where we used Ruby for defining steps in our approach. Even if one step is pending, the entire scenario is marked as pending. In case the first step fails to execute or is pending, the entire scenario is skipped. Step definitions can be used and re-used and one step can be called from another.

We have used the framework Calabash to support automated tests written in ruby and Gherkin. It contains all the scenarios in the featurefile written in natural language, Gherkin. It also contains the step definitions which act as a server side code which are written in Ruby language. Calabash Android provides the client-server architecture. We have included many custom defined functions which are used in our current approach.

4. Case Studies

To apply the developed tool for dynamic mobile application testing, we used several realistic mobile applications and system. NDTV, Waze, and WordPress APP are selected for the study objects. We created step definitions for multiple apps to implement different functionalities and scenarios. The operational flow to set up the calabash android and finally run the test scripts is as below.

Step 1: To define the calabash android environment;
Step 2: To set up the *apk* parameters to build;
Step3: To obtain the *apk* and start the calabash console;
Step 4: To generate a feature directory;
Step 5: To run cucumber;
Step 6: Run the calabash android;
Step7: Calabash Android Dem Spec Created.

4.1 Study Results

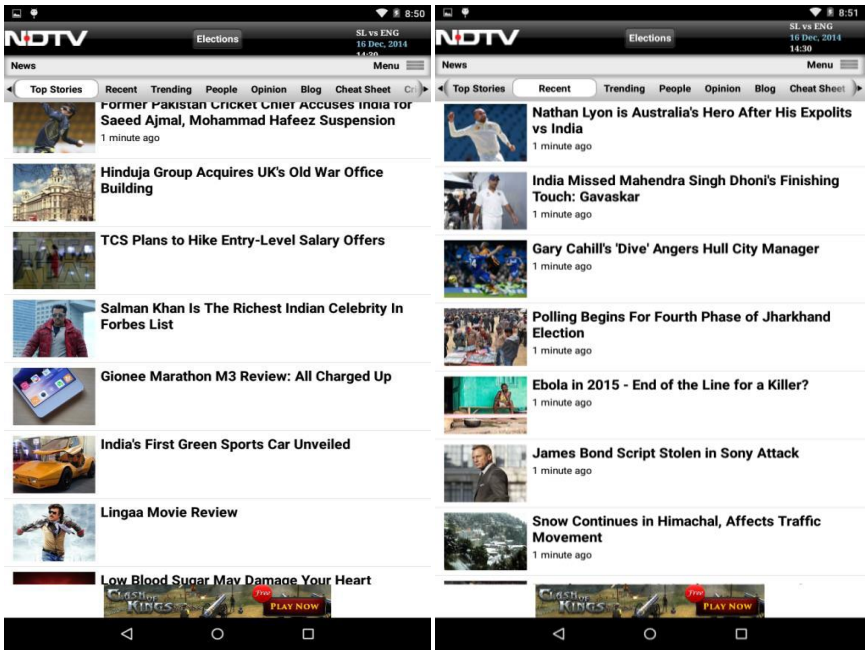
Case 1: NDTV App- Syntax

Initially we started working on the NDTV new app for testing the buttons and scrolling features of the app.

We defined multiple scenarios where a test engineer can test different properties according to the requirements. Here we explicitly defined 5 different scenarios. In the first scenario we change the headline once the app is launched. When the “news” shows, we define the steps to scroll down. As recent headlines appear, we further scroll down and use the button press feature to select the “recent” button on the app. Then we wait for progress and again scroll down. Yet again, we press the “trending” button and scroll down. In this way, we can create multi-level scenarios for testing different attributes of the application. We have used the NDTV *apk* file from the Google Play and tested it using the scenarios. When we try to run the calabash android gen command, it creates a sub directory.

When all the cases are tested successfully, it displays the results as passed including the number of successful scenarios. The scenarios are executed on an android device and the screenshots depict the execution summary for different scenarios for NDTV app. Figure 5 presents the test results when running the scripts for the app. Right below NDTV we can see the text “News” which will be read by the application as a starter sign to execute next steps in the scenario shown in Figure 5(a). The next steps are directing the application to scroll down the page three times. The second screenshot shown in Figure 5(b) here shows the same feature of the script file but a different scenario. Here the second tab after “Top Stories” is clicked which can be seen down below as “Recent”.

DOI reference number: 10.18293/SEKE2017-170



(a) (b)

Figure 5 Test results when running the scripts for 'NDTV' app

Case2: WAZE App- User Sign in Syntax

We further enhanced the functionalities of our approach and implemented user *sign-in* feature for waze app as depicted below. The scenario is to log into the application. The scenario is defined as below.

The test engineer logs into the application and upon login should see the User Agreement and then touch the Accept check box. Next user touches the Login button and enters the user id and password and then presses sign-in. Figure 6 shows the sample of syntax for testing the scenarios.

my_first-6.feature

```
Feature: To test a map based android app

Scenario:  To enter the app


Given I wait for 3 seconds
And I see the text "OUR AGREEMENT"
When I scroll down
And I scroll down
And I scroll down
Then I press view with id "settingsCheckboxText"
And I touch the "Accept" text

And I see the text "Welcome to Waze!"
And I touch the "Login" text
And I touch the "Existing users sign in:" text
And I enter text "lpshikhar" into field with id "userName"
And I enter text "carboncs15" into field with id "password"
And I press view with id "signin"
```

Figure 6 A sample syntax for testing the scenarios

Case 3: Wordpress App: Scenarios and Syntax

The next step was to execute some key functionalities for WordPress App. The below execution summary depicts the different scenarios for this app. Two scenarios are defined in detail. The first scenario is to log into the app being a valid user. The test engineer can validate the sign-in feature of app using this natural language based scenario. As the user gets to sign-in, it enters the username and password and then presses the sign-in button. As the user enters the application, it should see the "Posts" tab and press view for "more options". The user then logs out of the application. There is a two second timer after which the user is prompted to sign-in again.

The second scenario describes the scenario when the login information is incorrect. The user should get a message that "login information is not correct". Figure 7 presents the feature of two scenarios.

my_first-3.feature

```
Feature: Login feature

Scenario: As a valid user I can log into my app
    When I see "Sign in"
    Then I enter text "lpshikhar" into field with id "nux_username"
    And I enter text "shikharrawat" into field with id "nux_password"
    Then I press "Sign in"
    And I should see "Posts"
    And I press view with content description "More options"
    And I touch the "Sign out" text
    And I press the "Sign out" button
    Then I wait for 2 seconds
    And I should see "Sign in"
    Then I enter text "lpshikhar" into field with id "nux_username"
    And I enter text "shikharrawat" into field with id "nux_password"
    And take picture

Scenario: Wrong log in info
    When I see "Sign in"
    Then I enter text "lpshikhar" into field with id "nux_username"
    And I enter text "shikhar" into field with id "nux_password"
    Then I press "Sign in"
    And I should see "The username or password you entered is incorrect"
```

Figure 7 A sample syntax for testing the scenarios

4.2 Study Limitations

While designing the system and developing the tool, we faced several issues. Some of those issues are listed below.

We had used emulator for testing the apps which took

DOI reference number: 10.18293/SEKE2017-170

longer time to run the whole scenario. To improve this we replaced the emulator with the android tablet we have and reduced the running time. Regarding to the resign tool, when the apps are tested, the *.apk* file needs to use the same signature in our approach. To achieve this we used the *re-sign* jar which is a drag and drop java tool that strips the *apk* file from its signature and then signs it with our android debug key and solves the issue.

5 Conclusions and Future Work

As more constructions and deployments of mobile APPs and web applications on devices, engineers need more quality validation research and test automation tools to deal with the related issues and challenges. Currently, there is a need for an autonomous test scripting architecture for mobile apps, which we focus in this paper. This paper is a step forward towards natural language scripting. The intent of this work is to limit the dependency on any one scripting language and come out with the universal widely defined natural language.

The proposed approach has a lot of application scope in the future industry. There is also a scope for automation in the future as the present server side code requires manual testing. This approach suffers from some limitations. For example, the syntax is currently dependent on variables. Thus if the variables associated change, the step definitions need to be modified. The new id's need to be extracted from the app properties which is a complex problem in the known space. Therefore, there is tremendous scope to automate and reduce this dependency on the variables in the future work.

References

[1] H. van der Merwe et al., "Verifying Android Applications Using JavaPathFinder," ACM SIGSOFT Software Eng. Notes, vol. 37, no. 6, 2012,pp. 1–5.

[2] R. Mahmood et al., "A Whitebox Approach for Automated SecurityTesting of Android Applications on the Cloud," Proc. 7th Int'l WorkshopAutomation of Software Test (AST 12), 2012, pp. 22–28.

[3] J. Bo et al., "MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices," Proc. 2nd Int'l Workshop on Automationof Software Test (AST 07), 2007, pp. 8–14.

[4] S. Anand et al., "Automated Concolic Testing of Smartphone Apps,"Proc. ACM SIGSOFT 20th Int'l Symp. Foundations of Software Eng. (FSE12), 2012, pp. 1–11.

[5] D. Amalfitano et al., "Using GUI Ripping for Automated Testing ofAndroid Applications," Proc. 27th IEEE/ACM Int'l Conf. Automated SoftwareEng. (ASE 12), 2012, pp. 258–261.

[6] T. Kallio and A. Kaikkonen, "Usability Testing of Mobile Applications:A Comparison between Laboratory and Field Testing," J. UsabilityStudies, vol. 1, no. 1, 2005, pp. 4–16.

[7] R. Mizouni et al., "Performance Evaluation of Mobile Web Services,"Proc. 9th IEEE European Conf. Web Services (ECOWS 11), 2011,pp. 184–191.

[8] T. Puhakka and M. Palola, "Towards Automating Testing of CommunicationalB3G Applications," Proc. 3rd Int'l Conf. Mobile Technology,Applications & Systems, 2006, article no. 27, pp. 1–6.

[9] I. Satoh, "Software Testing for Wireless Mobile Computing," IEEE WirelessComm., vol. 11 , no. 5, 2004, pp. 58–64.

[10] H. Song et al., "An Integrated Test Automation Framework for Testingon Heterogeneous Mobile Platforms," Proc. 1st ACIS Int'l Symp. Softwareand Network Eng., 2011, pp.141–145.

[11] E. Giordano et al., "MoViT: The Mobile Network Virtualized Testbed,"Proc. 9th ACM Int'l Workshop Vehicular Inter-networking, Systems, andApplications, 2012, pp. 3–12.

[12] C. Q. Tao, J. Gao, "Modeling Mobile Application Test Platform and Environment: Testing Criteria and Complexity Analysis", Proc. the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing, 2014, pp 28-33.

[13] R. Buyya, et al., "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities", Proceedings of International Conference on the High Performance Computing & Simulation (HPCS), 2009, pp 1-11.

[14] W.T. Tsai, Y. Hang, and Q. H. Shao, "Testing the Scalability of SaaS Applications", Proc. IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2011, pp 1-4.

[15] W. Hargassner, et al.,"A Script-Based Testbed for Mobile Software Frameworks", Proc. of International Conference on Software Testing, Verification, and Validation, 2008, pp 448-457.

[16] I. Satoh, "A Testing Framework for Mobile Computing Software". IEEE Transaction on Software Engineering, Vol. 29(12), 2012, pp 1112-1121.

[17] H. Muccini, A. D. Francesco, and P. Esposito, "Software Testing of Mobile Applications: Challenges and Future Research Directions", Proc. International Workshop on Automatic Software Test Automation, 2012, pp 29-35.

[18] J. Gao, et al., "Mobile Application Testing: A Tutorial", IEEE Computer, 47(2), 2013, pp 46-55.

[19] J. Hartikainen, (July, 2013) Why use user story based testing Retrieved (July, 2014) from http://codeutopia.net/blog/2013/07/28/why-use-user-story-based-testing-tools-like- cucumber-instead-of-other-tddbdd-tools/.

[20] C. Siemens, (November 2013) The Search of Mobile App Test Automation. Retrieved (June, 2014) from http://engineering.zillow.com/the-search-for-mobile-app-test-automation/.

[21] Testing Methods and Tools. Retrieved (July, 2014) from http://www.methodsandtools.com/tools/tools.php

[22] B. Phar, Hooking into the Test Process. Retrieved (July, 2014) from http://docs.behat.org/guides/3.hooks.html

[23] TDD Style of software development Retrieved(Aug, 2014) from www.ibm.com

Acknowledgement

This paper is supported by the National Natural Science Foundation of China under Grant No.61402229, 61502233, and 61602267; the Open Fund of the State Key Laboratory for Novel Software Technology (KFKT2015B10); the Postdoctoral Fund of Jiangsu Province under Grant No.1401043B, and the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant no. 15KJB52003.

DOI reference number: 10.18293/SEKE2017-170