2012 2nd International Conference on Biotechnology and Environment Management
IPCBEE vol.42 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCBEE. 2012. V42. 20

Model-based Automated GUI Testing For Android Web Application Frameworks

Sutasinee Methong [+]

Software Systems Engineering Program, The Sirindhorn International Thai-German Graduate School of Engineering, King Mongkut's University of Technology North Bangkok, Bangkok, Thailand

Abstract. The mobile phones today have become a vital part of everyone's life. Since the very first mobile phone appeared in the 80s until launching of the smartphones with high resolution display touch screen that enable multi-touch gestures. However, in the perspective of software or applications, though there are over a hundred thousand applications available to download but for the consideration of openness, user control, and customizability due to the fact that Android powers hundreds of devices whereas the users can experiment and customize their phone as they need. The aim is to study and improve the manual testing approaches by adapting the automated testing framework based on Model-based testing approach and applied with the Heidelberg Mobil International GmbH web-application on Android browser, and then the results will display the differences in time and effort of test executed and productivity between manual testing and automated testing.

Keywords: Model-based Testing, Automated Testing, Web Application, Android, Selenium, WebDriver, AndroidDriver, Page Object, etc.

1. Introduction

As number of smartphones continue to increase, the abundance of mobile applications and their abilities. Therefore, the expectation towards the mobile application is not only how they meet user demand or accuracy, but also the appearances, response and interaction with the Graphical User Interfaces (GUI) [1] allow human interacting with the application, and it is essential to ensure that the functions beneath its appearances are working as promises and as expected. Hence, the traditional software testing techniques are most likely not thoroughly addressed the testing requirements and coverage criteria to test the mobile applications. This study targeted at the challenges of testing the mobile applications.

HDM-I or Heidelberg Mobil International GmbH [2] is a German company, with over 12 years of expertise in the range of mobile services and in developing online information systems for pedestrians, which focusing on location-based systems and on intuitive Human-Computer interaction. The company has developed the applications and portals for cities, businesses, events, trade show, and exhibitions which the main goal is to offer a user friendly, and mobile location-based information system, by providing a cross-platform application like web-based application, while the former procedures for mobile web application development had been evolving functional testing and non-functional testing and into the level of unit testing, which used to be done by the developers, but for the user interface and usability process have been done manually by the Quality Assurance team, the development along with number of applications lead to the frustration for QA team to performing tasks before the applications are launched to clients.

Though manual testing allow tester to perform more ad-hoc testing or random testing [3], which can unveil defects and bugs but it is error prone, also time-consuming and tedious task that require heavy human effort investment of human efforts and resources as mention earlier and more importantly it is nearly

[+] Corresponding author.
E-mail address: s.methong@gmail.com.

impossible for human to cover all the branches and criteria, including when the applications, or its functions need to be re-tested after bug fixed.

This paper present an adaptation of Automated GUI testing on Android Mobile Web Application frameworks, which is organized as follows: The Model-based testing for Platform independent system design, System Under Test (SUT), GUI Automated testing with Selenium 2.0 or WebDriver, Test Case consideration and the used of Page Objects, an application of Automated GUI Testing by implemented Eclipse Plug-in development, and then presents the results, conclusion and future works.

2. The Model-Based Testing For Platform Independent System Design

Model-based testing [4] step in and assist the problem of complexity in order to have the system tested in abstraction level, which correspondent with concept on how the software is developed such as the object oriented, whit the high-level programming languages. Though Model-based testing is the application of Model based design that use for designing and optimally executing the necessary artifacts for performing software testing. As will be discussed further regarding the Software under test (SUT), the model describing the SUT, which is the target software or system component which will be tested. Additionally, the model is usually an abstract or partial presentation of the SUT's desired behavior. Accordingly the test cases will be derived from the particular model will be the functional tests, whereas the test is on the exactly same level of abstraction as the model.

3. System Under Test (SUT)

3.1. Android Mobile Web Application

For Google's Android operating system, the web application [5] can be accessed through android web browser, which Android providing web application developers by:

- Support for view-port properties which enable the properly size the web application based on the screen size
- Enable CSS and JavaScript features for developers for having the different styles and images based on the screen's pixel density (resolution)

3.2. CeBIT2go Mobile Web Application

The key factors for Automated Testing are to identify and understand about System under test or SUT. By identifying what software is to be tested, which specific components and features, also the environment surrounding the SUT. The case study of CeBIT2go mobile web application which developed by HDM-I by delivering exhibitions information not only „Just in Time" but also „Just in Place" to users. Whenever users enter the exhibitions, with mobile ticketing is enable. Also when visiting particular exhibitors, and purchase an order, the mobile payment is activated which optimally adapted to the environment as shown in figure 1.

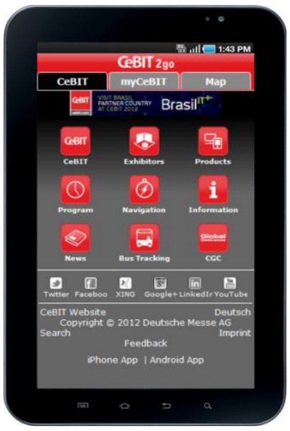
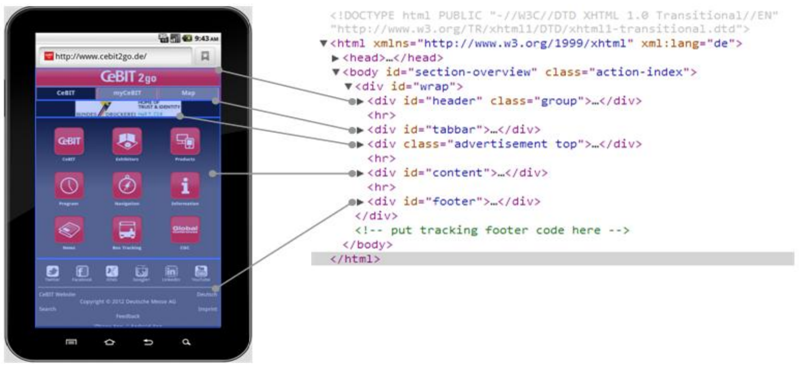


Fig. 1: GUI of CeBIT2go mobile web application on Samsung's Galaxy Tab.

3.3. User Interface Elements

The user interfaces elements which CeBIT2go provides users are an object as button, image icon, list, or open area which user can freely move the map around with their fingers when the devices is enable the multi-touch. However, what is behind the GUI is the Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS). The HTML provides the structure of the page, together with the CSS which is the layout. Though, CeBIT web application also need the graphics and scripting while the HTML and CSS are the basis to build the web application. The briefs HTML structure of the CeBIT2go web application can be represented as below in figure 2.



```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
▼<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
  ▶<head>…</head>
  ▼<body id="section-overview" class="action-index">
    ▼<div id="wrap">
      ▶<div id="header" class="group">…</div>
       <hr>
      ▶<div id="tabbar">…</div>
      ▶<div class="advertisement top">…</div>
       <hr>
      ▶<div id="content">…</div>
       <hr>
      ▶<div id="footer">…</div>
     </div>
     <!-- put tracking footer code here -->
   </body>
  </html>
```

Fig. 2: HTML Structure of CeBIT2go mapping with User interface elements.

3.4. Selenium 2.0 or WebDriver

Selenium is browser automation frameworks and also open-source software under Apache 2.0 license. Selenium primarily for automating web applications for testing purposes, also enabling the exercise of web based administration tasks automatically.

Selenium 2.0 has been improved from with the main features is the integration of WebDriver API. WebDriver or Remote WebDriver is the crucial interface where the test will be written against and enable user to drive the browser constitutionally whether locally or on remote machine by using Selenium Server using WebDriver as a client implementation. Selenium offers various Drivers from WebDriver for supporting the advance web applications testing, which have various supports on different browsers.

3.5. Android Driver[6]

AndroidDriver is a driver for running on Android devices or emulator, AndroidDriver uses Remote WebDriver as mention in above section. In Server-client architecture, while the test code is on the client side and with the prior installation of the server or android-server-2.6.0.apk onto the emulator or device. The application executes the test and display as Android WebView, with the embedded of a light-weight HTTP server or Jetty and the communication between client and server is done via Wire Protocol (which is REST based and requests by using JSON over the HTTP).

4. Test Case Design and Consideration

The study presented in this paper examines the behavior of application under test, which can be extracted to three main test considerations. Additionally according to the testing goal which based on its model and behavior can be shown as follows: (1) Verify Page Elements: Check if UI elements such as Page Title, Header, Tab bars, Icons, Button etc are present once the page is loaded successfully. (2) Page transformation: Once the click is initiated and page changes, or load new page, and check if we are on the right page (3) Functionality Testing: Design as user scenario to test specific function.

4.1. Page Object

Page Object is Design Pattern where Objects Representing the Page are Exposed to the Test Logic as Services Providing Access to the Element and Logic of the Page. Which it can be Defined as

- The public methods exposes the services which the page offers

- The internals of the page should not be exposed
- No assertions in Page Object class
- Which the assertions will be in the test class as test's logic
- Only represent "Service that need to be tested"

The benefits of applying Page Object are consolidating the code for interacting with any given UI element, encourage Code reusability, also reduce duplication, improve maintainability, which made the test more readable, robust and less brittle. Additionally, it can hide detail of telling the browser the "How to do".

4.2. Plug-in Development

Once the test scripts are ready, and to assist HDM-I QA team with "Automatically testing approach" Therefore the Eclipse plug-in is implemented which the requirement of the plug-in as follow:

- Install Android WebDriver APK
- Port forwarding TCP:8080
- Launch the test suite automatically
- Generate Test result which consist of HTML report, Test log in XML and Text document and Screen shots

As a result, the plug-in [7] can be installed in another machine from menu Window > Show View > "HDM-I Automated Test View", and provided users with

- Though the emulator need to be set up and started
- And make sure that the computer detect both emulator and device, which all devices will be shown in DBMS panel
- The test suite in Package Explorer panel
- WebDriver will be install automatically, which replace the normal steps to set up the test such as:
  - Install WebDriver.apk
  - Start Android WebDriver application though UI of device by click on icon or via command prompt
  - forwarding via terminal using command line
    $./adb -s <serialId> forward tcp:8080 tcp:8080
- Launch the test from selected devices for this example the devices used are emulator and Galaxy tab
- While the test is running the progress bar is also running

5. Result

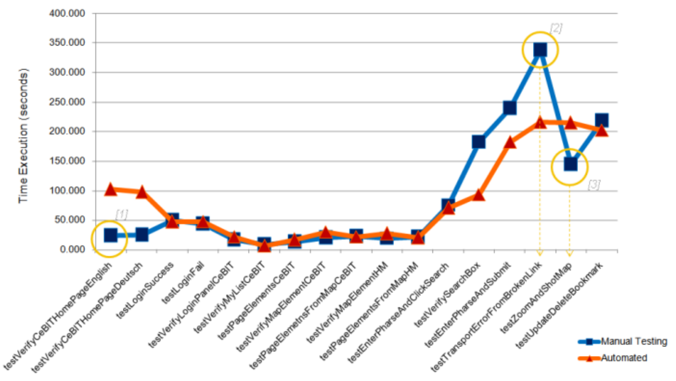


Fig. 3: Result of Time execution compared between Manual and Automated Testing.

Comparison between time spent on testing 17 test cases on Samsung Galaxy Tab: Manually and Automatically The experiments set up by (1) Automated Testing: Execute All test cases with no errors (2)Manually Testing: Each test case performed manually with the same test case specification and step according to the step in test scripts. While the time capture from start initial action onto device and stop timer after last action performed. The result can be discuss into detail as follows:

1. The first noticable point where Automated testing spend more time compare to Automated testing, once we consider the test code which consist of: Loading page, Locate UI element,

Assertion if the element is met expectation. While Manually testing, once the page is loaded. Tester can only scan though the application page.

2. Test case of "testTransportErrorFromBrokenLink", this test case designed for testing the link to access the Map in the application. Automated Testing: Take much less time, as the link can be send and insert directly from test script to web browser, while Manual testing: Tester need to edit and insert sample link to browser manually By having the multiple of incorrect link.
3. Test case of "testZoomAndShot Map", This test case designed for exercise the zoom in/out function of Map together with the capture screen shot after action perform as the interaction between Tester and Map using test script is difficult to determine whether the cases is met expectation therefore the screenshot is applied. By take the screenshot after every zoom so the time spent was more than Zoom in/out manually without captures any screen. Therefore the judgment of zoom function is works expectedly will be determined by tester.

6. Discussion

After the study shown how automated testing can be applied for Android mobile web application frameworks, therefore the question of what are we looking for in Automated Testing raised. Although In real world, A thousand of test suites are running everyday on the large web system Though Not so many new bugs are found but there are 2 main factors which are:

- In order to free up the valuable time that tester would spend on regression testing for Exploratory Testing
- Enable the opportunity to heavily refractor or make major change quickly and confidently

As a conclusion, Testing against HDM-I venue web application automatically can benefits not only one single Application/Projects as long as the GUI and Web interfaces are similar not by the appearances but by the structure of HTML page source. The major benefits of applying Automated Testing to the usual quality assurance process as (1) Test cases can be executed fast, and implementing is fun and challenging, which require analytical &reasoning skills. (2) Consider of reusability, maintainability and represent contribution for future use. (3) Test coverage and scopes can be varied (4) one can test: Unit (class/method/functions), a Module or System**.**

7. Acknowledgements

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study. First and foremost, my utmost gratitude to Prof. Dr. rer. nat. Horst Lichter, thesis advisor at RWTH Aachen University and Dr. Matthias Joest, Head of Developer at Heidelberg Mobil International whose sincerity and encouragement. Miss Rebekka Reichbauer has been my inspiration as I hurdle all the obstacles in the completion this research work. Moreover I would like to acknowledge the quality assurance team, developers at HDM-I and colleagues at HDM-I for providing me information, help and friendship. Last but not the least, my family, my father who is my role model in living the meaningful life, my mother for supporting me in every steps of the way and my brother who is always be with me.

8. References

[1] H.Thimbleby, **User Interface Design**, ACM Press, New York, NY, 1995

[2] D.Hoffman, **Test Automation Architecture: Planning for Test Automation**, Software Qulity Methods, LLC, CA, 1999

[3] Heidelberg Mobil International GmbH (HDM-I), **About HDM-I**, http://www.heidelberg-mobil.com

[4] S.R.Dalal, A.Jain, N.Karunanithi, et al., **Model-Based Testing in Practice**, Bellcore, NJ, May 1999

[5] Android Developer, **Web Apps Overview**, http://developer.android.com/guide/webapps/index.html

[6] SeleniumHQ, **AdroidDriver**, http://code.google.com/p/selenium/wiki/AndroidDriver

[7] E. Clayberg, D. Rubel, **Eclipse Building Commercial-Quality Plug-ins second edition**, Addison-Wesley, Boston, MA, 2006