Volume 5 Issue VI, June 2017
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Automated Testing for the Graphical User Interface of Android Applications Using an Open Source Mobile Testing Tool Appium

Vinay. S. Pattanshetti[1], Mr. Ramesh C N[2]
[1]Dept. of MCA, Siddaganga Institute of Technology Tumakuru, Karnatak
[2]Asst. Professor, Dept. of MCA, Siddaganga Institute of Technology, Tumakuru, Karnataka

***Abstract:** this project mainly deals with an approach on how to do automated test case generation of the appearance of responsive web page which helps to test the display of the same web page on different device screens. The introduction of mobile devices with smaller screens motivates the need for web pages that work correctly across many different devices referred to as responsive web design. Mobile access is a key feature for companies: both to reach new customers, and also to provide an enhanced service to existing customers. Testing the correct appearance of a responsive web page on different devices is not a trivial task because there are no standard rules for responsiveness, and the layout may need to be significantly rearranged in order to fit on smaller screens.*

I. INTRODUCTION

Testing is an integral part of software engineering and preoccupies significant prerogative in product-based industry applications. Today, companies are spending increasing amount of time and resources in ensuring the application is fully tested for best user experience and optimum performance by the application, by manual testing. Testing is carried out to make sure that daily updates are reflected correctly. In this paper, automated testing for mobile applications, both hybrid and web are discussed. This paper comprises of the technology used in testing the Graphical User Interface of Android applications using an open source mobile testing tool Appium. By replacing the manual testing by automated testing using scripting techniques can improve the effort per unit time and the accuracy per test case. Furthermore, a comparison between Appium and other automated testing tools is done to state the advantages of using Appium for mobile User Interface testing.

Software testing is an integral part of software development process. Software testing is analyzing a system or a component by providing defined inputs and comparing them with the desired outputs to check the discrepancies between the desired and actual outputs and correct them. Basically software testing can be divided into two categories.

A. Manual testing & Automated software testing

Manual software testing is as the name suggests done manually that it requires human input, analysis and evaluation. Automated software testing is the automated version of manual software testing.

Software testing has evolved since 1970's as an integral part of software development process. The reason being the ability to check the errors and faults present in the software so that they are corrected. Testing is done in many phases depending upon the requirements of the software being developed. Testing can be both manual and automated depending on what suits the requirements. Testing is a planned process with care taken especially on what test has to be done when. We will be discussing automated software testing in detail in this paper.

II. AUTOMATION TESTING

A. Definition of automation testing

Using automated software testing helps in avoiding the errors humans make. The error may occur due when humans get tired of doing the process repeatedly. Automated testing programs will not do the same they will not miss a test by mistake. The automated test program will also provide a means of storing the results of the tests accurately. The results can be automatically fed into a database and can be used to provide useful statistics on how the software development process is progressing.

B. Need of Automation testing

1) Effective Smoke (or Build Verification) Testing: Whenever a new software build or release is received, a test (generally referred to as "smoke test" or "shakedown test") is run to verify if the build is testable for a bigger testing effort and major application

Volume 5 Issue VI, June 2017
ISSN: 2321-9653

functionalities are working correctly. Many times we spend hours doing this only to discover that a faulty software build resulted in all the testing efforts going to waste. Testing has to now start all over again after release of a new build. If the smoke test is automated, the smoke test scripts can be run by developers to verify the build quality before being released to the testing team.

2) Standalone - Lights Out Testing: Automated testing tools can be programmed to kick off a script at a specific time. If needed, automated tests can be automatically kicked off overnight, and the testers can analyses the results of the automated test the next morning. This will save valuable test execution time for the testers.

3) Increased Repeatability: At times it becomes impossible to reproduce a defect which was found during manual testing. Key reason for this could be that the tester forgot which combinations of test steps led to the error message; hence, he is unable to reproduce the defect. Automated testing scripts take the guess work out of test repeatability.

4) esters can focus on Advanced Issues: As tests are automated, automated scripts can be base-lined and re-run for regression testing. Regression tests generally yield fewer new defects as opposed to testing newly developed features. So, functional testers can focus on analyzing and testing newer or more complex areas that have the potential for most of the defects while automated test scripts can be used for regression test execution.

5) Higher Functional Test Coverage: With automated testing a large number of data combinations can be tested which might not be practically feasible with manual testing. We use the term 'Data driven testing' which means validating numerous test data combinations using one automated script.

6) Other Benefits

a) Reliable: Tests perform precisely the same operations each time they are run, thereby eliminating human error.

b) Repeatable: You can test how the software reacts under repeated execution of the same operations.

c) Programmable: You can program sophisticated tests that bring out hidden information from the application.

d) Comprehensive: You can build a suite of tests that cover every feature in your application.

e) Reusable: You can re-use tests on different versions of an application, even if the user-interface changes.

f) Better Quality Software: Because you can run more tests in less time with fewer resources.

III. MANUAL TESTING V/S AUTOMATION TESTING

Manual Testing	Automation Testing
Manual Testing is Time consuming and tedious: Since test cases are executed by human resources so it is very slow and tedious.	Automated Testing is Fast and runs test cases significantly faster than human resources.
Manual Testing is a huge investment in human resources: As test cases need to be executed manually so more testers are required in manual testing.	Automated Testing is a less investment in human resources: Test cases are executed by using automation tool so fewer testers are required in automation testing.
Manual Testing is performed where the user operations of the application are performed manually without the use of any automated tools.	Automated Testing is performed where the user operations of the application are performed automatically by running some scripts using automated tools.

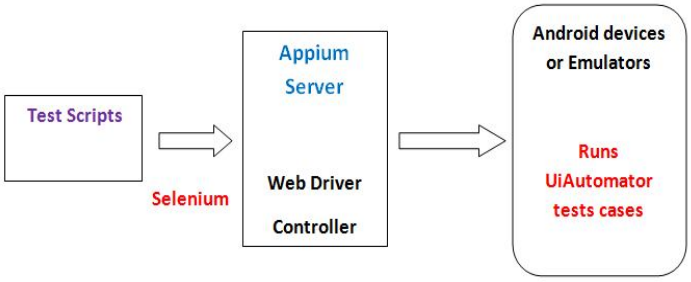
Volume 5 Issue VI, June 2017
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

IV. ARCHITECTURE OF APPIUM

Appium finds its roots from the Selenium WebDrivers, which is the standard for browser automation, used by developers in large numbers for programming in any language of their choice. Appium uses the Selenium WebDrivers to execute scripts. And being an open source cross platform tool, scripts can be written in Java, Ruby, C, Python, Perl which gives the programmers the liberty to use any language.

This section describes the architecture pertaining to Android platforms. Following fig. shows the framework for Android apps.

Fig. The Appium framework for Android apps



The architecture of Appium can be divided into four major components;

A. *Web Drivers scripts*

Selenium WebDriver libraries and APIs is used to write the test case scripts. These scripts are similar for both Android and iOS for a particular event of a similar application.

B. *Appium Server*

Appium Server is an HTTP server which handles and creates the Web Driver sessions. It starts a test case execution that spawns a server.

C. *Instruments or UiAutomator*

Appium server listens to proxies' commands, Instruments Command Server for iOS apps and UiAutomator running on device for Android apps.

D. *Real devices or simulators and emulators*

A simulator is the replica of a real iOS device which is used for testing the app. Similarly, an emulator is for Android. Appium executes the test scripts relatively faster on a real device, without knowing the technicality of the application code, making it just the ideal framework desired.

V. WORKING OF APPIUM

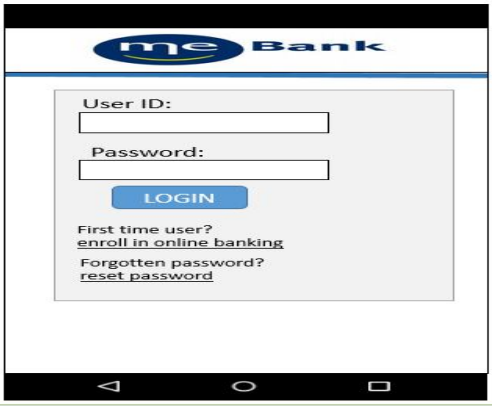
The scripts and the desired capabilities can be written in any programming language including Python, Perl, Java, Ruby, C#. Here, an eclipse IDE is used to write capabilities and scripts both on Windows and Android platform. An example of desired capabilities is shown in the following code. These are the capabilities for Android app being tested on the emulator. For an iOS app, just the desired capabilities need to be changed, for instance, device name will become iPhone and platform name will become iOS.

```
File appDir=new File ("directory_location");

File app=new File (appDir,"sample_apk_name.apk");

DesiredCapabilities cap=new DesiredCapabilities();

cap.setCapability (CapabilityType.BROWSER_NAME,"");

cap.setCapability ("deviceName","AndroidEmulator");

cap.setCapability ("platformName","Android");

cap.setCapability ("app", app.getAbsolutePath());

RemoteWebDriver driver=new RemoteWebDriver (new URL
("http://127.0.0.1:4722/wd/hub"), cap);

System.out.println("Testing");
```

Volume 5 Issue VI, June 2017
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

A. Finding elments in UserInterface

1) *By Xpath:* Xpath is an abstract representation of a path of an element. In the following figure, the elements for username and password are found by navigating to a particular element on the screen forming a hierarchy. Following shows the script demonstrating the usage of Xpath. For entering a username, the Xpath used is driver.FindElement (By.XPath(<xpath query expression>)) By ID: The Appium inspector recognizes the id of a particular UI element. This id is used to locate the element while generating the script.
2) *By Name:* The element is recognized by its name. In the following Fig., the Login button is recognized by the name Login.



VI. TEST CASES AND RESULTS

A. Sample test case

1) Verify the horizontal spacing of account section in account overview page in tile view.
2) Horizontal spacing between tile view and list view icons is 10 pixels.
3) Accounts are displayed as list under account section.
4) Horizontal spacing between right margin and account section is 10 pixels.

TestRun Id 2	
Passed	11
Failed	4
Inclusive	0
Skipped	0
Total	15

Class Name
TransferMoneyPageTest

Test Name	Status	Duration (Sec)	Failure/Ignored Message	stack-trace
AmountLinkTap_TestCase	Pass	87.936349		
DeleteLinkTap_TestCase	Pass	71.329405		
DescriptionDetails_TestCase	Pass	42.556559		
ErrorMessageDisplay_TestCase	Pass	37.412039		
HorizontalSpacingFields_TestCase	Fail	38.498096	Pixel difference between datetextbox and date picker icon is not as expected pixel difference. Expected: Same But was: Difference	at Fiserv.DigitalChannels.Corillian.Test.Responsive.Emulation.Portrait.TransferMoney.PageTest.HorizontalSpacingFields_TestCase() in c:\Sify\e_copy-updated\2.1-2-17\product\automation\Responsive\Emulation\Portrait\TransferMoney\PageTest.cs:line 65
HorizontalSpacingRecurringFields_TestCase	Pass	38.696635		
LayoutFieldsWithoutRecurring_TestCase	Pass	32.896098		
MakeRecurringLink_TestCase	Pass	35.541786		

B. NUnit Framework

NUnit is a unit-testing system for .net applications in which the whole application is segregated into assorted modules. Every module is tried autonomously to guarantee that the goal is met. The NUnit framework cooks a scope of qualities that are utilized amid unit tests. They are utilized to characterize test - fixtures, test strategies, expected exception and ignore techniques.

NUnit is an open source item. You can get this instrument alongside source code and change the source code if required for your particular needs. In any case, 99% of the software engineers don't disturb the source code.

NUnit gives graphical UI application to execute all the test scripts and demonstrate the outcome as a win/disappointment report (rather than message boxes from the test script). NUnit is the most well known unit testing system for .net applications.

VII. CONCLUSION

My contribution is primarily the novel idea of consistency. The idea of consistency is defined in terms of mapping between the master web page and the one that we are comparing against the master web page. Using this idea we could automate entire process of generation of automated test cases for responsive web pages. The pages are checked for consistency by applying rules which a user can activate by selecting checkboxes or eventually putting one's own rules by writing the script. In the work I showed that the basic idea is valid with the help of implemented tool with four high level rules. The developed program demonstrates how the current approach works.

The working time was distributed between exploring the topic with related materials and tools and the development part. All the sections were completed within the determined time boundaries. Hence, from the organizational point of view, the project is successful.

REFERENCES

[1] Burnetein(2003), Practical Software Testing: process oriented approach, Springer Professional Computing. P. Ammann and J. Offutt (2008), Introduction to Software Testing, New York J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[2] Cambridge University Press. K. Karhu, T. Repo and K. Smolander(2009), Empirical Observations on Software Testing Automation, International Conference on Software Testing Verification and Validation. Milad Hanna, Mostafa Sami, Nahla El- Haggar(2014),K. Elissa, "Title of paper if known," unpublished.

[3] A Review Of Scripting Techniques Used in Automated Software Testing, International Journal of Advanced Computer Science and Applications, Vol. 5, No. 1. Swati Hajela(2012), Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[4] http://www.guru99.com/appium

[5] Appium Cook Book

