International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161

Available at http://inpressco.com/category/ijcet

General Article

Automated Testing of Mobile Applications using Scripting Technique: A Study on Appium

Shiwangi Singh[Å*], Rucha Gadgil[Å] and Ayushi Chudgor[Å]

[Å]Information Technology, Plot No. U-15, J.V.P.D Scheme, Bhaktivedanta Swami Marg, Vile Parle (West), Mumbai - 400056, India

Accepted 15 Oct 2014, Available online25 Oct 2014, **Vol.4, No.5 (Oct 2014)**

Abstract

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 and iOS 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.

Keywords: *Scripting techniques, mobile application, Appium, Automated testing, UI testing, iOS apps, Android apps*

1. Introduction

According to Ilene Burnstein, software testing is described as a group of procedures carried out to evaluate some aspect of a piece of software (I. Burnetein, 2003). In addition, software testing means evaluating software by observing its execution (P. Ammann and J. Offutt, 2008). Software testing is broadly divided into two main categories: manual and automation testing. Manual testing is defined as the process of testing the system under test manually. In this process, the tester interacts with the system posing as the end user and interacts with different functionality present in the system to eliminate all possibility of errors. In automation testing, we make use of software different from system being test to control the execution of tests. Here, the output generated is compared to predefined output to ensure the correct working of system.

Software testing encompasses majority of software development life cycle making it a costly process to carry out. Thus, the challenge lying ahead of us is how to reduce these efforts while maintaining the overall quality of the system being developed.

Thus, automation is the most prominent and efficient solution to this challenge, as it reduces the cost of the entire process by incorporating it into testing of repetitive tasks makes it quicker and efficient than manual testing.

2. Manual Testing vs Automated Testing

Software testing can be classified into two categories mainly as manual testing, and automated testing and can be distinguished as follows:

In manual testing, no programming can be done to write sophisticated tests which fetch hidden errors and points to missing information while in automation testing testers can program sophisticated tests to bring out hidden errors and points to missing information.

Secondly, manual testing requires huge investment in human resources but is also less reliable as tests may not be performed with precision each time because of human errors. This type of testing becomes slow, tedious and time consuming. With the advent of increasing number of mobile applications, Automation provides us with a solution , by performing same operation each time they are run eliminating risk of errors as this process makes use of scripts and allowing us to test more number of possibilities to reduce further errors.

Katja Karhu summarizes the difference between the two categories by suggesting that automated software testing should be used to prevent new errors in the already tested working modules, while manual testing is better used for finding new and unexpected errors (K. Karhu, T. Repo and K. Smolander, 2009). The two approaches are complementary to each other, automated testing can perform a large number of test cases in little time, whereas manual testing uses the knowledge of the tester to target testing to the parts of the system that are assumed to be more error-prone.

Section 3 is divided into further sub-sections delineating the Types of Data Centric Applications in section 3.1, Test case cardinality in section 3.2, Architecture of Appium in section 3.3, Working of Appium in section 3.4.

3. UI Automation Testing

*Corresponding author: **Shiwangi Singh**

Over the years, mobile technology has made rapid advancements leading to changes in the business processes and IT solutions. This has made it imperative to revise the software testing techniques, including test design, test automation, performance testing and UI testing. In this paper, a study of UI automation testing of Mobile applications is done so as to show its impact factor in the field of automation testing. Moreover, there is a need for a tool which could test mobile applications across platforms. Today, the most popular mobile applications are based on Android and iOS platforms. The functionalities of the iOS and Android apps can be successfully tested using Appium automation tool.

3.1 Types of Data Centric Applications

Data Centric Applications can be of the following three types, Native Apps, Mobile Web Apps and Hybrid Apps(Swati Hajela, 2012). Appium supports all these three types of applications thus enabling the testing of modern applications which generally uses the browsers or web today.

1. Native Apps: These apps are both installed and launched on mobile devices. Testing of such applications on some mobile devices requires access to a device ID. The functionality and usability of native apps need to be tested on multiple devices.
2. Mobile Web Apps: These apps do not require installation. They are required to be tested on varied mobile browsers.
3. Hybrid Apps: These apps are a combination of a native app and mobile web, where the icon and interface of the native app is merged with the ease and the rich content of the mobile web.

3.2 Cardinality

The basic idea of Appium is the generation of scripts of an event to be tested. The scripts (i) can be hard coded, (ii) can have a control structure, (iii) can be both data-driven and can have a control structure [4]. Thus scripting techniques can be used to automate the UI testing. So, in this method, the script remains the same, but the input test cases can be multiple. Consider the cardinality between the number of test cases and the script to be run as shown in the fig 1. Test script recorded for one particular event can be run as many times as the number of test cases to be validated.

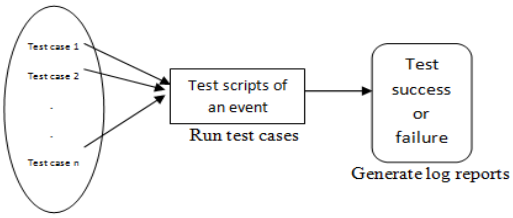

Fig.1 Test cases to script cardinality

3.3 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 both Android and iOS platforms. Fig.2 shows the framework for iOS apps and fig.3 shows the framework for Android apps.

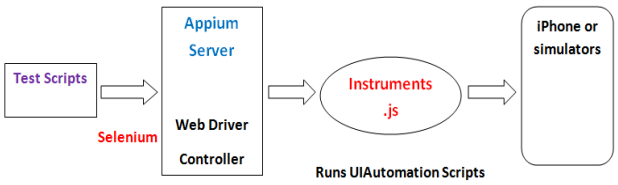

Fig.2 The Appium framework for iOS apps

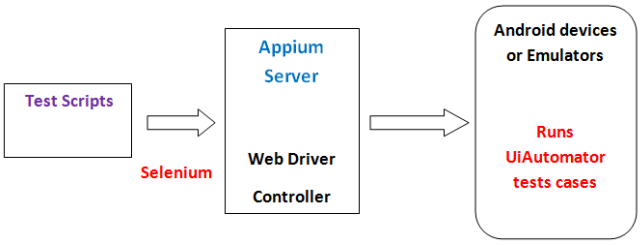

Fig.3 The Appium framework for Android apps

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.

3.4 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 platform. An example of desired capabilities is shown in the Fig.4. 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");
```

Fig.4 Desired Capabilities for Android app testing

While testing the GUI of the applications, the elements are identified automatically by Appium inspector including the element name, type, Xpath and class. Xpath is finding the element by the path with which that particular element is navigated to. Appium provides the facility to record this script and directly use them while running the test case. A specific recording of an event will have a unique script for that specific path. Here, in Fig.4 and Fig.7 the test cases are written in Junit. Appium Inspector captures a screenshot of the application screen. Record will generate the script for the command you click or tap on the screen. When a tap or a click command is given, the following screenshot of the next screen can be captured. A script for this command is then subsequently recorded. The whole process of capturing, recording and generating scripts is demonstrated is as shown in Fig.5.

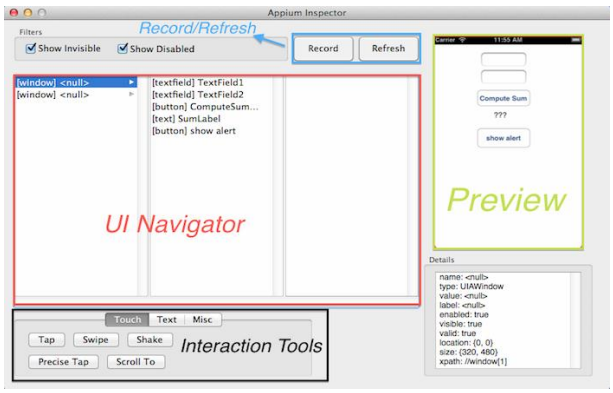

Fig.5 Overview of Appium Inspector[]

So, in this way a hard coded script for a particular event is generated. Now, this raw script can be used as a data-driven script by connecting it to an external data file and providing it a control structure in a program running the test cases.

Coming to the very important property of automated testing, that is, reusability, can be achieved using appium. The same set of scripts can be used for testing a particular test case against hundreds of values. The effort and the time required to test manually with a hundred values will require hours of work, but with appium it can be completed in minutes automatically.

Finding an element in a UI screenshot

By Xpath: Xpath is an abstract representation of a path of an element. In the Fig.5 the elements for username and password is found by navigating to a particular element on the screen forming a hierarchy. Fig.7 shows the script demonstrating the usage of Xpath. For entering a username, the Xpath used is, *android.view.View[1]/android.widget.ListView[1]/android.view.View[1]*. Similarly, for entering a password, only the *android*.view.View[2] changes in the Xpath.

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.

By Name: The element is recognized by its name. In the Fig.6, the Login button is recognized by the name Login.

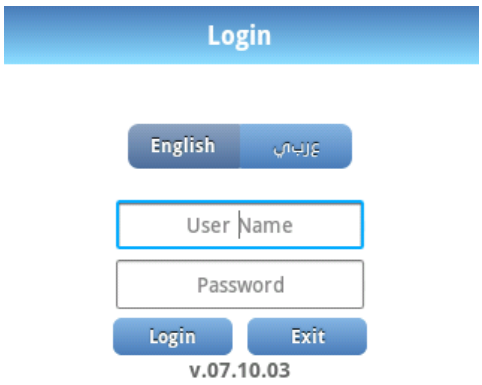

Fig.6 Login screen of an Android app

Issuing a Command to the Appium Server

Events that can be tested using Appium: Selenium Web Drivers allows the usage of various methods for testing the UI elements. Selenium Web Driver recognizes a command from the code and sends it in the form of Json via a HTTP request to the Appium Server. UI functional testing of events such as click, adding texts, tap and scroll is supported by Appium.

Click event: Elements.click() event is used to send a click command to Appium, which interfaces with the client device and executes the command. After its execution, a log report is sent to the Appium Server. Thereafter Appium is ready to interface further command.

Tap event: A tap event is like a click event but it is used to tap anywhere on the UI to get to a particular screen or tap in a textbox just before sending any key onto the textbox.

sendKeys event: A piece of text called keys is sent from the code to be typed onto the textbox in the client device.

Table 1 Comparison between Appium and other Automation Tools

Automation Tools	Platform Support	Open source cross platform	Same scripts running on different platforms	Browser support
Instruments	iOS	No	No	Safari, Chrome
UiAutomator	Android	No	No	IE, Firefox
Selenium	iOS and android	Yes	Yes	Safari, Crome, Firefox, Opera, IE
Monkey Talk	iOS and Android	No	Yes	Firefox
Robotium	Android	No	No	Any html based
Appium	iOS and Android	Yes	Yes	Chrome, safari, Firefox, open support for all browsers

This process is similar to a user typing some text in a textbox. In Fig.5 a key "username" is sent by using Elements.sendKeys("username") to the empty username textbox.

Scroll event: A scroll event is supported by various methods to scroll up or down the screen.

There are some limitations as well pertaining to event support, for instance, testing of a pinching event is not supported by Appium.

```
driver.findElement(By.xpath("//android.view.View[1]/android.widget.ListView[1]/android.view.View[1]")).sendKeys("username");

driver.findElement(By.xpath("//android.view.View[1]/android.widget.ListView[1]/android.view.View[2]/android.widget.EditText[1]")).sendKeys("password");

driver.findElement(By.name("Login")).click();

System.out.println("Successfully logged in!");
```

Fig.7 Recorded script for the Login screen

4. Comparison

Comparison is shown in table 1

Conclusions

For industrial applications, the task of UI automation testing for mobile applications can be a tedious process when performed manually. Automation tool simplifies this task substantially reusing the scripts once recorded. Being an open source tool, Appium supports different platforms for mobile applications, namely android and iOS apps. It makes a tester's job easier since Appium can be used on both Windows and Mac OS . Appium allows applications to be tested without recompiling it or even without knowing the actual application code. Compared to other automation tools, Appium stands out in terms of platform independence, support for hybrid and web applications and supports various languages such as Ruby, Python and C#.

References

Burnetein(2003), Practical Software Testing: process oriented approach, *Springer Professional Computing*.

P. Ammann and J. Offutt (2008), Introduction to Software Testing, *New York: 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), 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), Automation Testing in Mobile Applications, *International Software Testing Conference*. http://appium.io/slate/en/master/?ruby# http://appium.io/slate/en/v1.0.0/#example