Proc. of the Intl. Conf. on Advances In Computing, Electronics and Electrical Technology - CEET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-005-7 doi: 10.15224/ 978-1-63248-005-7-52

# Android Native Mobile Application Functional Test Automation

Calabash-Android Automated Testing Technology

[Loke Mun Sei & Fairuz Mariman]

***Abstract*—Mobile application has become very popular and widely used. Hence, the need and importance of mobile application testing has significantly increased before the mobile application gets to go to market on time and within budget. This paper discusses on the challenges faced in testing a native mobile application, and how Calabash-Android automated testing technology is used to help in automating Android mobile application testing. The sample illustrated in this paper is running on a mobile device emulator.**

***Keywords*—test automation, mobile application testing, Calabash, Android, native application, emulator**

## I. Introduction

In 2013, there were over 1,790,000 mobile applications available in apps stores and the number of downloads reached 102 billion with total revenue of $26 billion [1] [2]. The mobile landscape is rapidly evolving, demanding mobile apps developers to deliver intuitive, reliable, and robust apps in a shorter time-to-market cycle and within budget while not compromising on the product's quality. The product has to be rigorously tested to ensure it is working in terms of its functionality and usability.

A good test strategy outlines the test approach to verify the product's functional and non-functional features. A comprehensive mobile application testing strategy should include a mixture of test devices (multiple versions of operating systems), types of network environment (Wi-Fi or cellular network), and the types of testing involved (Functional, Usability, Performance or Security). However, it is inevitable that test engineers will face challenges in mobile application testing. As a result, new trends in software testing have emerged for example "crowdsourcing", "test in the cloud" and "mobile application test automation" to overcome the challenges faced by test engineers in testing mobile applications.

Loke Mun Sei
Product Quality & Reliability Engineering, MIMOS Berhad
Malaysia

Fairuz Mariman
Product Quality & Reliability Engineering, MIMOS Berhad
Malaysia

## II. The Challenges

The following are the analysis of the challenges faced in the test phase of a native mobile application:

### A. *Device Diversity*

The biggest challenge when it comes to mobile application testing is the device variation. Mobile devices may vary from operating system version, manufacturer, screen resolution, memory, hardware, etc. Hence, it is costly and not feasible to test the mobile application on each and every of the real mobile devices available in the market. One common way to overcome this issue of testing on numerous physical devices is to use simulators or emulators. Emulators are easier to manage and cost-effective compared to the real devices.

### B. *Operating Personnel Human Error*

The functional test of mobile applications often involves manual tasks. The test engineer creates test cases and executes the test cases manually, step by step and indicates whether a particular step was accomplished successfully or whether it failed. It is very much depends on the individual's domain knowledge and testing skill. One test engineer may approach and perform a test differently than another, thus, operating personnel human error can occur easily if it all done manually.

### C. *Test Cycle Time*

During the functional test phase of a native mobile application, there might be several test iterations involved. Test engineers execute the test cases and report the test result. If the test result is failed, developers will fix the reported issue and test engineers will then retest and verify the same test cases. It is time consuming if the same test cases were to be manually executed in each and every of the test iterations and also during the regression test phase.

Since the allocated time for testing is limited and the mobile application needs to be published to the market within budget and time, it is usually impossible for test engineers to retest all of the existing test cases. The usual workaround would be to prioritize and to select and test a small subset of the existing test cases based on the timeline available. As a result, the test coverage would be reduced and the risk of defects escaping during the test phase could be alarming and very high.

Proc. of the Intl. Conf. on Advances In Computing, Electronics and Electrical Technology - CEET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-005-7 doi: 10.15224/ 978-1-63248-005-7-52

## III. The Solution

A test automation tool with the objective to address the challenges mentioned earlier while maintaining the quality of the native mobile application is needed. Hence, we use an automated testing technology, Calabash-Android to automate our mobile application test cases. The automated test cases can then be used as regression test to determine the new patch or version released does not affect or break the existing mobile application functions.

Calabash is an automated testing framework based on Cucumber, supporting both Android and iOS native applications. Cucumber is a tool that executes plain-text functional descriptions as automated tests. The text is written in a business-readable domain-specific language and serves as documentation, automated tests and development-aid that are all rolled into one format [3].

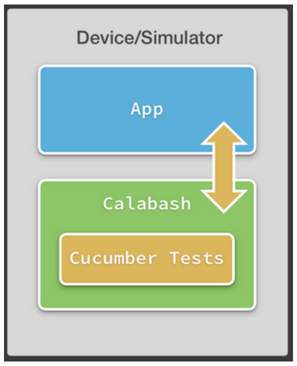


Figure 1. Calabash Framework [4]

With comparison to the test automation tool for web application, Calabash could be compared to Selenium WebDriver. However, interacting with a web application from a laptop or desktop computer (using keyboard or mouse click) is different from interacting with a native application using a touch screen. Calabash provides APIs that are specialized to native applications running on touch screen devices.

Calabash consists of two libraries: Calabash-Android and Calabash-iOS. Calabash-Android is the automation and testing library for Android, and similarly Calabash-iOS is for iOS (we will cover and focus on Calabash-Android library in this paper).

Calabash-Android is the underlying low-level library that empowers the Cucumber tool to run automated functional tests on Android phones and tablets as well as on simulators. This low-level library enables Quality Assurance (QA), business staff and developers to work at a high level by writing tests in a natural language using the terms and concepts of their business domain [5].

In Android, Calabash follows the same Client-Server model, but instead of the server portion running inside the mobile application, it actually runs as a separate application, which has permission to instrument the mobile application.

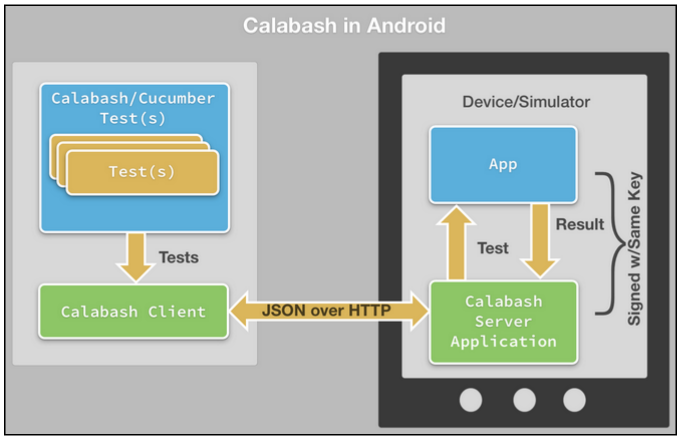


Figure 2. Calabash in Android [4]

### A. *Mobile Application Functional Test Automation Example*

This section will discuss on the steps by steps walkthrough on how the mobile application functional test is being automated. For example, we use the sample mobile application “apiDemos.apk” [6] and we want to automate the following test scenario:

- Launch “`API Demos`” application

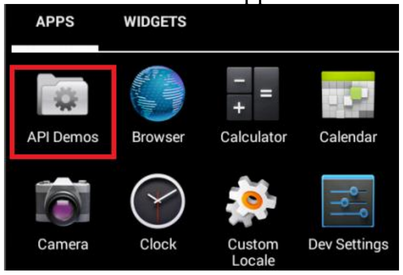


Figure 3. API Demos Application

- Press on “`App`” button

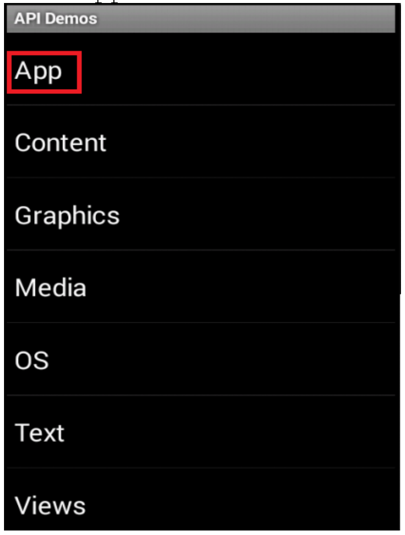


Figure 4. App Button

Proc. of the Intl. Conf. on Advances In Computing, Electronics and Electrical Technology - CEET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-005-7 doi: 10.15224/ 978-1-63248-005-7-52

- Wait for "`Activity`" button to appear and take a screen capture of the screen.

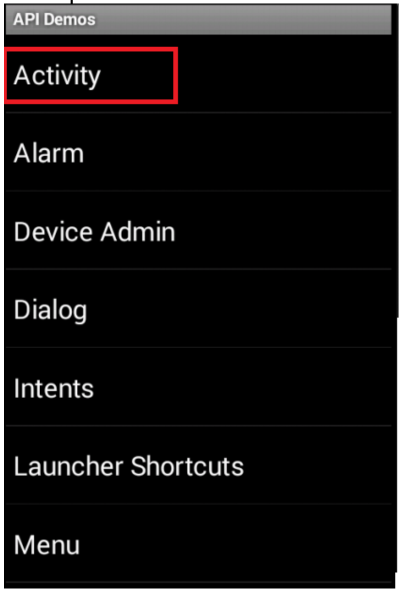
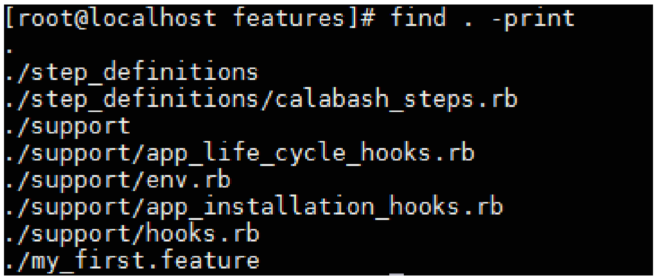


Figure 5. Activity Button

In order to automate the test scenario and test steps mentioned earlier, first, we need to create a Cucumber skeleton in the current folder using command "`calabash-android gen`". The directory structure of the Cucumber skeleton created is as shown in Figure 6.

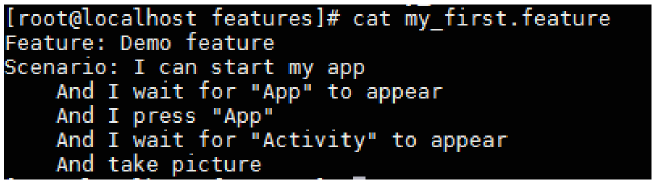


```
[root@localhost features]# find . -print
.
./step_definitions
./step_definitions/calabash_steps.rb
./support
./support/app_life_cycle_hooks.rb
./support/env.rb
./support/app_installation_hooks.rb
./support/hooks.rb
./my_first.feature
```

Figure 6. Cucumber Skeleton Directory Structure

To start creating the automation script, edit the feature file "`my_first.feature`" content to fit to the mobile application test scenarios.

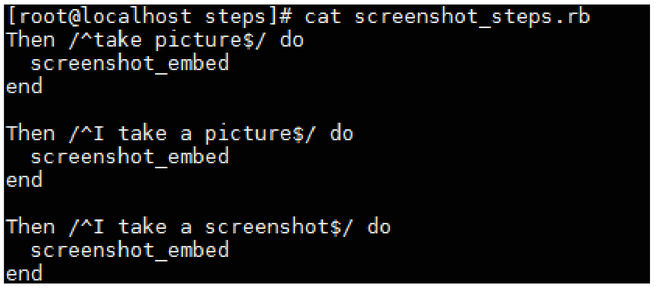
Calabash-Android has some predefined steps that can be called in a feature file by passing valid argument. In this case, we do not need to define these steps in the steps definition Ruby file. Figure 7 below is the sample feature file, "`my_first.feature`" to automate the test scenario and test steps mentioned earlier.



```
[root@localhost features]# cat my_first.feature
Feature: Demo feature
Scenario: I can start my app
    And I wait for "App" to appear
    And I press "App"
    And I wait for "Activity" to appear
    And take picture
```

Figure 7. Sample Automation Script

In this example, the predefined step "`take picture`" is being called. The "`take picture`" keyword is one of the predefined steps available in the steps definition file, "`screenshot_steps.rb`", i.e.



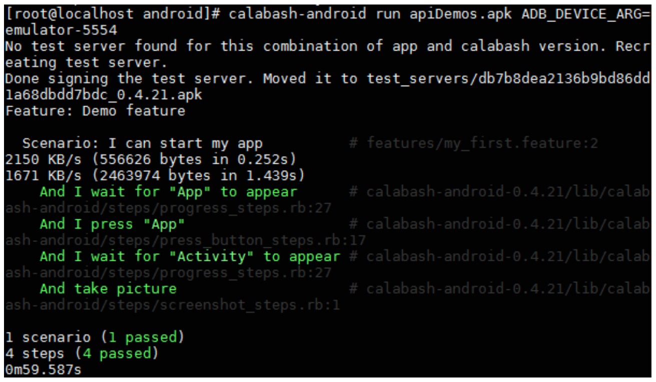
```
[root@localhost steps]# cat screenshot_steps.rb
Then /^take picture$/ do
  screenshot_embed
end

Then /^I take a picture$/ do
  screenshot_embed
end

Then /^I take a screenshot$/ do
  screenshot_embed
end
```

Figure 8. Predefined Steps Definition File – "screenshot_steps.rb"

## B. *Mobile Application Functional Test Automation Result*

Before the test automation script can be executed, a key store file needs to be created and then be used to resign the mobile application file. In this example, we will test the mobile application using android emulator. Hence, we start the android emulator before the automation test execution. Emulator allows us to switch device types and load a new profile, each time we select a device type. This gives us a preview of the real device where the mobile application would be downloaded.

After making sure the device is online, test automation script can then be executed with the command "`calabash-android run <apk_file>.apk ADB_DEVICE_ARG=<device_name>`". The automation script will be executed with test case status, i.e. green colour indicates the test is "Pass", and red colour indicates the test is "Fail", e.g. refer to Figure 9 for the test automation result. At the last step of the automation script execution, a screenshot will be taken and saved as "`screenshot_0.png`" in the system. The screenshot output will have the same capture as in Figure 5 mentioned earlier.



```
[root@localhost android]# calabash-android run apiDemos.apk ADB_DEVICE_ARG=
emulator-5554
No test server found for this combination of app and calabash version. Recr
eating test server.
Done signing the test server. Moved it to test_servers/db7b8dea2136b9bd86dd
1a68dbdd7bdc_0.4.21.apk
Feature: Demo feature

  Scenario: I can start my app          # features/my_first.feature:2
2150 KB/s (556626 bytes in 0.252s)
1671 KB/s (2463974 bytes in 1.439s)
    And I wait for "App" to appear      # calabash-android-0.4.21/lib/calab
ash-android/steps/progress_steps.rb:27
    And I press "App"                   # calabash-android-0.4.21/lib/calab
ash-android/steps/press_button_steps.rb:17
    And I wait for "Activity" to appear # calabash-android-0.4.21/lib/calab
ash-android/steps/progress_steps.rb:27
    And take picture                    # calabash-android-0.4.21/lib/calab
ash-android/steps/screenshot_steps.rb:1

1 scenario (1 passed)
4 steps (4 passed)
0m59.587s
```

Figure 9. Execute Test Automation Script and Result

Table I shows the estimated test execution time comparison for both manual and automated testing based on the sample test steps mentioned above. It has reduced approximately half

Proc. of the Intl. Conf. on Advances In Computing, Electronics and Electrical Technology - CEET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-005-7 doi: 10.15224/ 978-1-63248-005-7-52

of the test execution time by automating these simple test steps.

TABLE I. TEST EXECUTION TIME COMPARISON

| ***Test Step*** | ***Time Spent (second)*** | |
|---|---|---|
| | ***Manual*** | ***Automated*** |
| Copy apk file and install in mobile device/emulator | 60 | 60 |
| Launch "API Demos" application | 1 | |
| Press on "App" button | 1 | |
| Wait for "Activity" button to appear and take a screen capture of the screen (save the screen capture file) | 60 | |
| ***Total*** | 122 | 60 |

## IV. Conclusion and Future Work

From the study and research conducted, it clearly shows that this Calabash-Android automated testing technology has a great impact on the native mobile application functional testing as it reduces the test cycle time, reduces the cost of testing in real mobile devices and indirectly minimize human error with automated test steps.

For future work, we plan to create and add more commonly used test steps in the predefined steps definition file so that it can be shared across different mobile application projects. We also plan to extend the test automation to support iOS mobile testing by using the Calabash-iOS library.

## *References*

[1] Gartner Says Mobile App Stores Will See Annual Downloads Reach 102 Billion in 2013, http://www.gartner.com/newsroom/id/2592315

[2] Gartner: 102B App Store Downloads Globally In 2013, $26B In Sales, 17% From In-App Purchases, http://techcrunch.com/2013/09/19/gartner-102b-app-store-downloads-globally-in-2013-26b-in-sales-17-from-in-app-purchases/

[3] Cucumber - Making Behaviour Driven Development (BDD) fun, http://cukes.info/

[4] Introduction to Calabash, http://developer.xamarin.com/guides/testcloud/calabash/intro_to_calabash/

[5] Calabash: Functional Testing for Mobile Apps, http://blog.lesspainful.com/2012/03/07/Calabash/

[6] Sample mobile application file "apiDemos.apk" download, http://apk-recovery.googlecode.com/files/apiDemos.apk

About Author (s):

Loke Mun Sei
Senior Engineer for Product Quality & Reliability Engineering (PQRE) department, MIMOS Berhad. Focus mainly on test automation for both functional and security testing.

Fairuz Mariman
Lead for Functional Test Team of the Product Quality & Reliability Engineering (PQRE) department, MIMOS Berhad. Focus mainly on managing the team and to support the team with test automation idea.