International Journal of Latest Trends in Engineering and Technology
Vol.(8)Issue(2), pp.326-336
DOI: http://dx.doi.org/10.21172/1.82.044
e-ISSN:2278-621X

STRESS TESTING OF ANDROID APPLICATIONS USING TEST AUTOMATION

Santosh Kumar Sharma[1], Akshit Sharma[2] and Shivam Gaur[3]

Abstract- Android is an open source mobile operating system developed by Google and is based on Linux kernel. It is being used by major smartphone manufacturing companies. As the mobile applications and mobile consumers are rising swiftly and enormously, it is a thing of worry to researchers and testing professionals to concoct feasible testing procedures to guarantee the unswerving quality of these mobile applications. It is crucial to ensure the reliability of the applications running on these phones. Due to smaller development lifecycle of mobile applications, the developed apps tend to be flawed as little effort is put in ensuring the quality of the app. Thus rigorous testing is required for assuring its quality and that too in little time. But because of certain characteristics of mobile applications which are diverse from traditional applications, the traditional life cycle models cannot be incorporated into mobile application development. There are numerous mobile application testing techniques used such as manual and automated techniques. But the emphasis is given to automated approach due to its diverse advantages. This paper presents a test case for carrying out automated stress testing of android applications. Our experimental results show the validity of the test cases.
Keywords – Mobile Applications, Software Testing, Manual Testing, Automation Testing, ADB Shell

I. INTRODUCTION

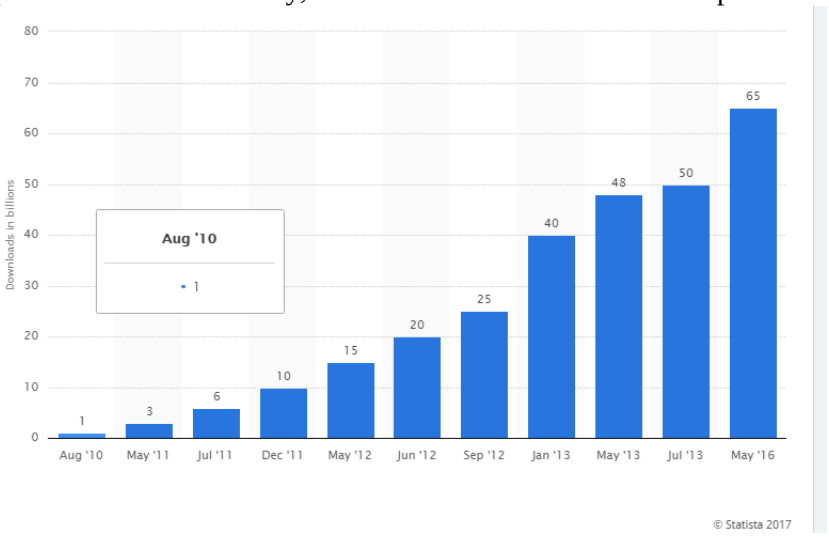
Mobile devices are taking over desktop computers at a fast pace and are becoming an essential part of our life. As the users of the smartphones are increasing at a rapid pace, so does the importance of application quality. It is estimated that by 2017 over 268 billion downloads of android applications will generate around $77 billion worth of revenue. Mobile app downloads are growing every year. So the testing of mobile applications needs to emphasize on functional testing, security testing, performance testing, usability testing, regression testing and compatibility testing. Testing is one of the important factors in improving application quality. The apps written for smartphones are increasingly becoming advanced and complex, adjusting to the constantly developing computational power of the hardware. The techniques used for testing desktop/laptop software are accommodated in testing mobile apps too.

Most of the previous studies focused on the testing of computer software, little thought has been given to the testing of mobile applications. In order to fill in the void on mobile application testing, this study tries to answer the following research questions:

- How to develop test scripts for automation?
- Does test automation enhance the effectiveness of mobile application testing?

[1] *Assistant Professor, Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi, Jaipur Campus, Jaipur, Rajasthan, India*
[2] *B.E. Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi, Jaipur Campus, Jaipur,*
[3] *Rajasthan, India B.E. Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi, Jaipur Campus, Jaipur, Rajasthan, India*

The remaining paper is organized as follows. First, we study about software testing and android mobile application development in brief and then recommend a test case for automated stress testing. Moreover, we describe the experimental results. Finally, conclusions and discussions are reported.



The bar chart shows number of cumulative app downloads from Google Play Store from August 2010 to May 2016

II. BACKGROUND

A. *Software Testing*

Myers quoted that "testing is the process of executing a program with the intent of finding errors." Also, software testing is any activity "aimed at evaluating an attribute or capability of a program or system". It was urged that the testing is better to be performed by someone who is not a part of the development team. Moreover, using a good test case has a high probability of finding an unidentified error. Testing can be broadly classified into white-box testing and black-box testing. White-box testing is typically performed internally especially during the early stages of software development, while black-box testing is commonly conducted externally when the development is completed. The focus of the white-box testing is on the depth of testing. The aim of black-box testing is to affirm that the program works the way it is expected to work.

B. *Android Mobile Applications*

Android mobile applications' source code is written in Java in either Eclipse IDE or Android Studio provided by Google. Google released the first edition of Android Software Development Kit (SDK) in September 2008. The Software Development Kit compiles the application code, data, resource files, etc. and creates an archive package called Android Package (APK). This package file has a .apk extension. The APK of an android application can be installed and run on an Android operating system which is a Linux-based system to host different apps. Each application is treated as a separate user in the operating system and a unique ID is designated to the application when it is executed. All files associated with the application are restricted to the application based on the ID. Each application has its own virtual machine which is separated from other applications.

Android Debug Bridge (ADB) is a command line tool provided by the android SDK (Software Development Kit). It allows developers to communicate with a connected Android-powered device and

install APKs, run shell commands, store logs, and emulate touch actions from the screen. All this helps in automation testing. The shell commands can be used to determine whether an android-powered device is connected to a PC and count the number of devices connected to the PC if any. The shell commands can also be automated by running a script. Android Debug Bridge can be used for grey-box testing, which is a combination of white-box testing and black-box testing.

C. Challenges faced in android application testing

The mobile application testing has different sets of problems such as:

1) Fragmentation: There are multiple devices in the market with different versions of OS.
2) Testing on multiple devices: As there are numerous mobile devices, it is not feasible to get a new device every time and simulators are not dependent.
3) Time to market: Time to market is reduced greatly. Release cycles have become short and rapid.
4) Newer Versions: A new android version is released typically every 10-12 months.
5) Multitasking: Since mobile devices have small screen sizes, it is not easy to show multiple applications simultaneously. Multiple applications open and running in the background lead to a lot of battery consumption.
6) Form Factor: There are different types of mobile devices such as phones and tablets. Developing the applications for various forms is related but designing apps for them is very different and time-consuming.
7) Diversity: There is a vast diversity of android based handsets in terms of screen size, OEM, operators, etc.
8) Emulator vs. Real Device: Testing on emulators is cost effective as testing is quick and efficient. While testing on real android devices helps to understand the application activities in real-life setups.
9) Manual vs. Automated: In comparison to manual testing by the human software testers, automated testing is economic, quick and but it requires a large amount of initial capital.

D. Automated testing of mobile applications

As the development of applications goes through a very short cycle, it is necessary that testing life cycle should also be quick. Various automation techniques are embraced for testing. Automation testing helps to increase efficiency and scope of application for better updates.

Advantages of automated testing:

1) Saves time.
2) Identify bugs and defects.
3) Higher quality application developed.
4) Improves accuracy of the application.
5) Test cycles are quick.
6) Lowers the cost even though the initial cost is high but it is compensated with long term use.

E. Automated testing versus Manual testing

The automated testing technique is highly enticing and due to this reason, automated testing is capable of decreasing human errors and efficient in identifying bugs quickly. In fact, automated testing allows the tester to verify the critical features of the application by testing different data sets. According to Quilter, automated testing is capable of executing large volumes of repeating scenarios which are difficult to carry out manually.

Manual testing is very time-consuming in comparison to automated testing, and often it has limitations in testing through the finite user interface of the mobile device. Manual testing acknowledges the tester to create test cases and pursue the test case design and instruction design to obtain the specific test goals.

III. RESEARCH METHODOLOGY

The main goal of the study is the test case development for the android applications. In particular, we devised test case for stress testing of android applications.

The testing of mobile applications brings into light certain concerns and challenges. These include the type of device model, the version of the android operating system, the method to test compatibility, and the amount of testing required. Testing can be further divided into different levels: unit testing, integration testing, system testing, user-interface testing, regression testing, and acceptance testing.

The mobile applications are typically embedded to hardware devices which need to follow telecommunication regulations from different countries. The android applications on smartphones deal with various activities such as the interactions with different hardware components, communication with the operating system, memory management, and interactions with other applications and user interface. A comprehensive test case is necessary to test a particular mobile application. The analysis of logs generated by the test tool helps in identifying the bugs in the application.

Following the guidelines provided by GSM Association, MMS Conference Document Version 2.0.0, and CMMI for Software Engineering Version 1.1, we prepared test case for automated testing of mobile applications. In the test case, we clearly defined the test purpose, initial conditions, testing procedure, and expected behavior.

A. Test case design for automated stress testing

For automated stress testing, we prepared a test case which used audio, camcorder, and camera features of the mobile phone. The test was performed automatically. The Android APKs involved are the music player, camcorder, and camera.

Test Purpose

To ensure that the functionalities of the audio, camcorder and camera feature work properly while using and switching them in a resource constrained environment.

Initial Condition

1. The Android mobile phone is fully charged.
2. At least one music file is stored in the SD card.

Test Procedure (automatic procedure)

1. Load the audio player, load the music file and wait for five seconds. Play the music for 30 seconds.
2. Load the camera application and wait for five seconds.
3. Select the back camera and take a picture and wait for five seconds to save the picture.
4. With the front camera take a picture and wait to save the picture.
5. Switch to video mode and wait for five seconds. Using the back camera record a video clip of 30 seconds and wait for five seconds to save the video.
6. Switch to front camera. Record a video clip of 30 seconds and wait for five seconds to save the video.
7. Repeat the steps 1-6 1000 times.

Expected Behavior

1. While switching between the applications, there is no ANR (Application Not Responding) error.
2. The Audio file can be loaded and played correctly by the music application.

3. Images can be captured and saved correctly by camera application using both the front and back cameras.
4. Videos can be recorded and saved correctly by camera application. A video clip includes both audio and motion parts.
5. During the repetitive testing, the mobile phone will not reboot or crash.
6. Thumbnails of images and videos saved in media library can be loaded correctly.
7. Images and videos saved in the media library can be browsed and played correctly.

IV. Experiment And Outcome

A. Experimental environment

We used one branded mobile phone equipped with Android 4.4 and a 5-inch WVGA (800x600) screen. It has front and back cameras of 5 MP and 13 MP respectively. 8GB SD card was also inserted. While performing automated testing, a PC with Windows 10 was used.
For stress testing, we used ADB shell to emulate the touch screen controls for a series of stress tests. It was carried out 10 times.

B. Automatic testing configuration

Android Debug Bridge (ADB) shell was used for automatic control of the mobile phone and performing stress testing. After installing ADB shell and setting Windows environment variables, "adb device" command was used to ascertain the connection between the android mobile phone and PC over the USB cable. Following the procedure listed in stress test case, a script was made to call ADB commands and perform various tests. The following table shows the ADB shell commands used in the script. The script used is listed in the appendix. A batch file with a loop was used to execute the script 1000 times.

ADB Shell Commands Used

Command	Action
adb device	Lists the Android devices attached
adb logcat	Reads log of events
adb shell am start –n "APK"	Loads a particular APK application
adb shell getevent	Get event triggered from the touchscreen, Xand Y coordinate can be retrieved
adb shell input keyevent "value"	Triggers a particular event value=27: camera value=79: headsethook
adb shell sleep "SEC"	Pauses for SEC seconds

C. Stress test results

Based on the 10-run logs captured from ADB shell we did not encounter any error. It proved that the Android-powered phones passed the stress testing. ADB shell not only emulates the touch gesture but also sends system commands to load applications. Since ADB makes system calls to load APKs, it can capture potential errors occurred among the switching of applications if they share the same resources on multitasking.
In addition, each cycle took about 110 seconds to complete the execution. In order to finish the 10-run 1000-cycle tests, the automation process took over 300 hours. Although this is a relatively long-term testing, the testing in automation provided a reliable, efficient, and effective solution, compared to testing performed by human software engineers.

V. CONCLUSION

Since people wish for low-cost smartphones in low-income countries, Android-powered smartphones are being manufactured cost effectively to meet the increasing demands. However, the reliability and compatibility of such phones have been the major concerns of potential consumers. This paper instituted automated stress test case to address the concerns.

This study demonstrates the feasibility of carrying out automated testing on Android applications. Practitioners or software engineers may replicate the tests with minor changes to meet the requirements (e.g. different coordinate due to different screen sizes).

Future studies may acclimate our test case for other applications. Also, for the aim of generalizability, it is advised to replicate the study using different versions of operating system and different mobile handsets.

VI. APPENDIX

Android debug bridge test script

```
C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell am start -n
com.android.music/com.android.music.MediaPlaybackActivity
Starting: Intent { cmp=com.android.music/.MediaPlaybackActivity }

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Playing Music"
Playing Music

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell input keyevent79
Error: Unknown command: keyevent79
Usage: input [<source>] <command> [<arg>...]

The sources are:
    trackball
    joystick
    touchnavigation
    mouse
    keyboard
    gamepad
    touchpad
    dpad
    stylus
    touchscreen

The commands and default sources are:
    text <string> (Default: touchscreen)
    keyevent [--longpress] <key code number or name> ... (Default: keyboard)
    tap <x><y> (Default: touchscreen)
    swipe <x1><y1><x2><y2> [duration(ms)] (Default: touchscreen)
    press (Default: trackball)
    roll <dx><dy> (Default: trackball)

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 30

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event0 1 158 1

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event0 1 158 0
```

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell "am start -a android.media.action.IMAGE_CAPTURE"
Starting: Intent { act=android.media.action.IMAGE_CAPTURE }

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Enter Camera"
Enter Camera

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 42

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 42

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 42

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Picture Captured"
Picture Captured

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 293

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 35

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 293

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 35

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>echo -e "Switch Camera Sensor"
-e "Switch Camera Sensor"

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 42

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 42

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 42

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 465

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e " Sensor 2 Capture Picture Finish"
Sensor 2 Capture Picture Finish

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 180

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 463

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 180

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 463

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Enter Camcorder"
Enter Camcorder

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell input keyevent 27

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 30

C:\Users\sg2005\android-sdks\platform-tools>adb shell input keyevent 27

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Sensor 1 Record Video Finished"
Sensor 1 Record Video Finished

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 255

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 293

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 35

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 48 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 53 293

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 54 35

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 3 50 6

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 2 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event1 0 0 0

```
C:\Users\sg2005\android-sdks\platform-tools>echo -e "Switch Camera Sensor"
-e "Switch Camera Sensor"

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell input keyevent 27

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 30

C:\Users\sg2005\android-sdks\platform-tools>adb shell input keyevent 27

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Sensor 1 Record Video Finished"
Sensor 1 Record Video Finished

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event0 1 158 1

C:\Users\sg2005\android-sdks\platform-tools>adb shell sendevent /dev/input/event0 1 158 0

C:\Users\sg2005\android-sdks\platform-tools>adb shell sleep 5

C:\Users\sg2005\android-sdks\platform-tools>adb shell echo -e "Test Complete"
Test Complete

C:\Users\sg2005\android-sdks\platform-tools>
```

REFERENCES

[1] W. C. Hetzel and B. Hetzel, The Complete Guide to Software Testing, Wellesley, MA: QED Information Sciences, 1988.
[2] P. Sestoft, Systematic Software Testing, Version 2, 2008.
[3] Hsiu-Li Liao, Chen-Huei Chou, and Wan-Chun Chao, Functional Validation and Test Automation for Android Apps, International Journal of Machine Learning and Computing, Vol. 4, No. 6, December 2014.
[4] Bakhtiar M. Amen, Sardasht M. Mahmood and Joan Lu, MOBILE APPLICATION TESTING MATRIX AND CHALLENGES.
[5] Anureet Kaur, Asst Prof, Guru Nanak Dev University, Amritsar, Punjab, Review of Mobile Applications Testing with Automated Techniques, International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 10, October 2015.
[6] Software Engineering Practitioner's Approach By Roger S. Pressman.
[7] De Souza, Silva L., and de Aquino G.S., (2014), "Mobile Application Development: How to Estimate the Effort?", Computational Science and Its Applications–ICCSA 2014.Springer International Publishing, pp.63-72.
[8] Milano, D. (2001), Android Application Testing Guide. USA: Packt Publishing Ltd.
[9] Naik, S. and Tripathy, P. (2008) Software Testing and Quality Assurance: Theory and Practice. Hoboken: John Wiley & Sons, Inc.
[10] Selvam, R. (2011). 'Mobile Software Testing – Automated Test Case Design Strategies, International Journal on Computer Science and Engineering' (IJCSE) Vol.3

[11] CMMI Product Team. (2002). CMMI for Software Engineering, Version 1.1. Continuous Representation (CMMI-SW, V1.1, Continuous). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-2002-TR-028. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/02tr028.cfm
[12] CMMI Product Team. (2002). CMMI for Software Engineering, Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. Technical Report CMU/SEI-2002-TR-029. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/02tr029.cfm

[13] Myers, G., Badgett, T. and Sandler, C. (2012) The Art of Software Testing. (3red.). Hoboken, New Jersey: John Wiley & Sons, Inc.
[14] Selvam, R. (2011), Mobile Software Testing – Automated Test Case Design Strategies, International Journal on Computer Science and Engineering' (IJCSE) Vol.3.