A Novel GUI Testing Framework for Automated Testing of Android Applications

Dissertation

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY

IN

SOFTWARE TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SUBMITTED BY

ABHISHEK PANDEY

ROLL NO: 2K13/SWT/01

MAJOR PROJECT REPORT II

(Paper Code: CO 821)

Under the guidance of

Dr. Kapil Sharma

SHAHBAD DAULATPUR, MAIN BAWANA ROAD, NEW DELHI, DELHI

110042 INDIA

DELHI TECHNOLOGICAL UNIVERSITY

NEW DELHI

STUDENT DECLARATION

I hereby undertake and declare that this submission is my original work and to the best of my knowledge and believe, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of any Institute or other University of higher learning, except where due acknowledgement has been made in the text. Project work and published paper associated to the chapters are well discussed and improved under the guide supervision.

DATE:

SIGNATURE:

ABHISHEK PANDEY

ROLL NO: 2K13/SWT/01

DELHI TECHNOLOGICAL UNIVERSITY

NEW DELHI

CERTIFICATE

This is to certify that the thesis entitled "**A Novel GUI Testing Framework for Automated Testing of Android Applications**", is a bona fide work done by Mr. ABHISHEK PANDEY in partial fulfilment of requirements for the award of Master of Technology Degree in software technology at Delhi Technological University (New Delhi) is an authentic work carried out by him under my supervision and guidance. The matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma to the best of my knowledge.

DATE:

SIGNATURE:

Dr. Kapil Sharma

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

ACKNOWLEDGMENT

I am presenting my work on “A Novel GUI Testing Framework for Automated Testing of Android Applications” with a lot of pleasure and satisfaction. I take this opportunity to thank my supervisor, Dr. Kapil Sharma, for guiding me and providing me with all the facilities, which paved way to the successful completion of this work. His scholarly guidance and invaluable suggestions motivated me to complete my thesis work successfully. I am thankful to my friends and colleagues who have been a source of encouragement and inspiration throughout the duration of this thesis. I am also thankful to the SAMSUNG who has provided me opportunity to enrol in the M.Tech Programme and to gain knowledge through this programme. This curriculum provided me knowledge and opportunity to grow in various domains of computer science. Last but not least, I am thankful to all the faculty members who visited the Samsung premises to guide and teach. Their knowledge and efforts helped me to grow and learn in the field of computer science. This project has provided me knowledge in the area of Mobile Application Automation Testing and helped me in understanding the concept of accessing objects and designing framework for testing in Mobile Testing domain. I have given ample time and guidance to complete my project under timeline defined by the university..

ABHISHEK PANDEY

ROLLNO: 2K13/SWT/01

ABSTRACT

Model-based Automation Testing used to implement with Android that includes the way applications have been modeled, how tests were designed and execution has been done, kind of problems have been found in the tested application during the process. By evolving into mass market like products, smart phones and tablets created an increased need for specialized software engineering methods. To ensure high quality applications, constant and efficient testing is crucial in software development. However, testing mobile applications is still massive, time-consuming and error-prone. The mobile applications effective testing is an “emerging research area that faces a variety of challenges due to unique features of mobile devices”. Currently there are several studies and approaches regarding the testing of the functionality, security and usefulness of mobile application, still the field of GUI design testing has not been widely explored. This thesis concentrates on the design testing, which is the part of usability, though it focuses not on how well the users can interact with the application, but rather on the visual appearance of the GUI. The challenge for GUI testing is to verify whether native applications are correctly displayed on different devices. An automation of this testing can reduce time, effort, errors and the cost of the testing process, and increase productivity. Main principle which guided the project implementation were to implement the variability to elements such as environment, end points, test data, handling of test devices, core protocols, locations separated out from the test logic (scripts) and moved into hierarchical libraries for reusability and to prevent constant modification in whole test script every time UI changes in new devices.

CONTENTS

Table of Figures

Chapter One: **INTRODUCTION**

1.1 Scope & motivation:

Each and every product is tested by the software development group, but still the delivered software have always defects in it. All the test engineers try to detect these faults before releasing that product but these always edge in and appear again and again. These defaults creep in even when they are tested with the best processes of manual testing. To increase the effectiveness, efficiency and coverage of the software testing, automation is considered as the best way.

Human sitting in front of a computer is considered as the manual testing, who is carefully going through application and trying various input combinations and methods, comparing the results to the expected behaviour and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations. An automated testing tool is able to playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. Because of this, savvy managers have found that automated software testing is an essential component of successful development projects.

To ensure the quality of a software, software tests have to often be repeated during the development cycles, whenever the modification is made in the sousecode the software tests are repeated. For each release of the software it may be tested on all supported operating systems and hardware configurations. Manually repeating these tests is costly and time consuming. Once created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests. Automated software testing can reduce the time to run repetitive tests from days to hours. A time savings that translates directly into cost savings.

Importance of mobile applications are day by day increasing. In such a situation quality becomes the highest goal for the companies to provide the users these developed applications. Test engineers in the department of quality assurance of these companies are responsible for the defined level of software quality ensurance.

The mobile applications effective testing is an “emerging research area that faces a variety of challenges due to unique features of mobile devices”. Currently there are several studies and approaches regarding the testing of the functionality, security and usefulness of mobile application, still the field of GUI design testing has not been widely explored. This thesis concentrates on the design testing, which is the part of usability, though it focuses not on how well the users can interact with the application, but rather on the visual appearance of the GUI. The challenge for GUI testing is to verify whether native applications are correctly displayed on different devices. An automation of this testing can reduce time, effort, errors and the cost of the testing process, and increase productivity. The automation of the software testing process has numerous benefits, which are described by Melody Y. Ivory in “The State of the Art in Automating Usability Evaluation of User Interfaces”. The most important advantage is that of cost-saving, due to the reduction in the testing time. Another positive point is the prediction of the time and error expenses through the whole application. An additional Seite 9 benefit is the expansion of the tested features. The use of automated tools makes it possible to cover all possible test cases and user interactions, something that is not always achievable with non-automatic testing. Apart from this, the special tools not only perform automatically the test cases and simulate the user interaction with the system, but are also able to expertly analyze the obtained results. Not all testers have enough competence in all aspects of software evaluation. One more advantage is the possibility of the comparison between optional designs. During the manual testing only one designed UI is evaluated.

Some automated tools make it possible to predict and simulate the alternative and improved designs and to test them. Finally, the automated tests can also be performed during the development phase, with the UI schemes, prototypes and guidelines predicting

the bugs before implementing them. The human testers as a rule test only the implemented version of the UI.

Initially, most mobile applications were developed for entertainment purpose, but now many industries have arranged application development for competitive benefit. In order to support the image of the companies their applications should meet certain design requirements and follow the corporate identity.

The current situation shows that there is a need of an automated testing tool that can evaluate the visual design of the mobile application, according to the general design guidelines and the requirements of customers and the company. Such an automated design testing tool can improve and simplify the process of testing the user and corporate design requirements. The goal of this master thesis is to present approach for the Graphical User Interface automated testing for large scale. The result of this work is a prototype of the automated design testing tool for android applications, developed for all kind of applications whether UI Objects are available or not. The thesis includes an overview on existing systems and different methods of mobile testing, a general idea of a unified design testing tool for different devices, a presentation of the design testing approach with standardize scripting templates for large testing teams for better management of scripts for future use, testing and evaluation of this approach, as well as a discussion of the challenges and future research questions in the field of automated design testing of mobile applications.

1.2 Testing Automation:

The process in which the software tools execute prescripted testing on software application before it is released into production is called automated software testing.

The object of automated testing is to simplify as much of the testing effort as possible by using minimum lines of code. If there is huge percentage consumed by unit testing quality assurance team's resources, then this process might be a good candidate for automation testing. Tools of Automated testing are capable of executing test scenarios, reporting outcomes and comparing results with earlier test executions. Tests carried out with these tools can be run repeatedly and at any time.

The method which is used to implement automation is called a test automation framework. Many frameworks were implemented over the years by commercial vendors and testing companies. Automating tests with commercial or open source software can be complicated sometime, because they almost always require customization. Many organizations implement automation when it is determined that manual testing is not meeting expectation and bringing more human resources does not seems possible.

1.2.1 Automation:

The most well-known kind of application testing tool is automation, which attempts to replace human activities clicking and checking with a computer. The most common kind of test automation is driving the user interface, where a human records a series of actions and expected results. Two common kinds of user-interface automation are record/playback -- where a tool records the interactions and then automates them, expecting the same results -- and keyword-driven -- where the user interface elements, such as text boxes and submit buttons, are referred to by name. Keyword-driven tests are often created in a programming language, but they do not have to be; they can resemble a spreadsheet with element identifiers, commands, inputs and expected results.

Automation tools perform a series of preplanned scenarios with expected results, and either check exact screen regions -- in record/playback -- or only what they are told to specifically check for -- in keyword-driven. A computer will never say "that looks odd," never explore or get inspired by one test to have a new idea. Nor will a computer note that a "failure" is actually a change in the requirements. Instead, the test automation will log a failure and a human will have to look at the false failure, analyze it, recognize that it is not a bug and "fix" the test. This creates a maintenance burden. Test automation automates only the test execution and evaluation.

Another term for this kind of automation is something Michael Bolton and James Bach call checking -- a decision rules that can be interpreted by an algorithm as pass or fail. Computers can do this kind of work, and do it well. Having check automation run at the code level -- unit tests -- or user interface level can vastly improve quality and catch obvious errors quickly before a human even looks at the software.

1.2.2 Do you need automation?

Several of the teams we've worked with in the past have found themselves with a six-week test/fix/retest cycle. During that time, the technical staff was producing no new features. With three releases a year, the technical staff was testing 18 weeks -- which is more than a third of the time. Long retest cycles make rolling out experiments essentially impossible. Test automation is generally a natural fix for this; have the computer run automated checks, at least overnight, and you could release every day.

At the user interface (UI) level, there are many reasons to use automated tools for software testing. Preparing a small set of checks that runs frequently, building the system, verifying if major path of functionality fails and reporting the team on failure are all capabilities that automated testing can handle. This makes the feedback cycle important, so developers who introduced a major bug can find and resolve it quickly. Having these smoke tests in place can reduce the amount of effort the testers are spending on routine things, add confidence, and vastly reduce the cost of a test cycle without requiring years of automation work.

Once the automated checks are running at the GUI level, teams often find a different problem: their tests find too many bugs. When software is breaking too often, it's a sign that the team needs automated unit tests very small, technical tests at the code level that programmers can put in place. Programmers who run a unit test suite before check-in can prevent defects from escaping to the build.

The big problems test automation addresses are compressing test time, finding defects faster, and, in the case of unit tests, preventing regressions, where a feature may have worked a day ago but not after a new check-in. If the product tends to fail in ways that are different and unpredictable, or the UI is undergoing a massive change, checking the same things may have limited value. For example, a new UI that adds new required buttons will cause failures of the test suite because the button was not checked. If the success factors are less functionality and more usability -- if the product needs to be viral then focusing on test automation might not be the right approach. In these areas, where automation is not needed, direct interaction with humans is more important.

1.2.3 GUI and GUI Testing

There can be two types of interfaces in a computer application. First is Command Line Interface where you type text and computer responds to that command. Second is Graphical User Interface where you can interact with the computer system using images.

What is Graphical User Interface Testing?

GUI (Graphical User Interface) testing is the process of testing the system GUI of the System under Test. Graphical User Interface testing involves checking the UI with the controls like buttons, menus, icons, and different types of bars like tool menu bar, tool bar, windows and dialogue boxes etc.

Graphical User Interface testing is the process of ensuring functionality of the GUI for given application and making it conforms to its specifications.

Additionally GUI testing evaluates design elements like colors, layouts, fonts, font sizes, text boxes, labels, captions, text formatting, lists, icons, buttons, content and links.

1.2.4 GUI Testing Approach:

Graphical User Interface testing can be done via three ways:

1.2.4.1 Manual Testing

In this approach, screens are checked manually by testers for conformance with the requirements stated in software requirements specification document.

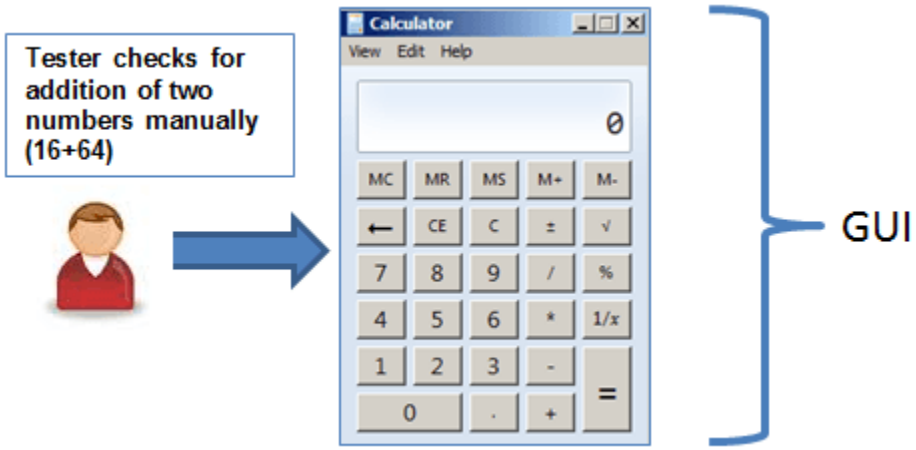


Figure 1 : Manual Testing

1.2.4.2 Record and Replay Automation

Graphical User Interface testing can be done using automation tools which used to done in two parts. In Recording process, steps are captured into the tool and in playback phase, recorded test steps are executed automatically on the Application under Test.

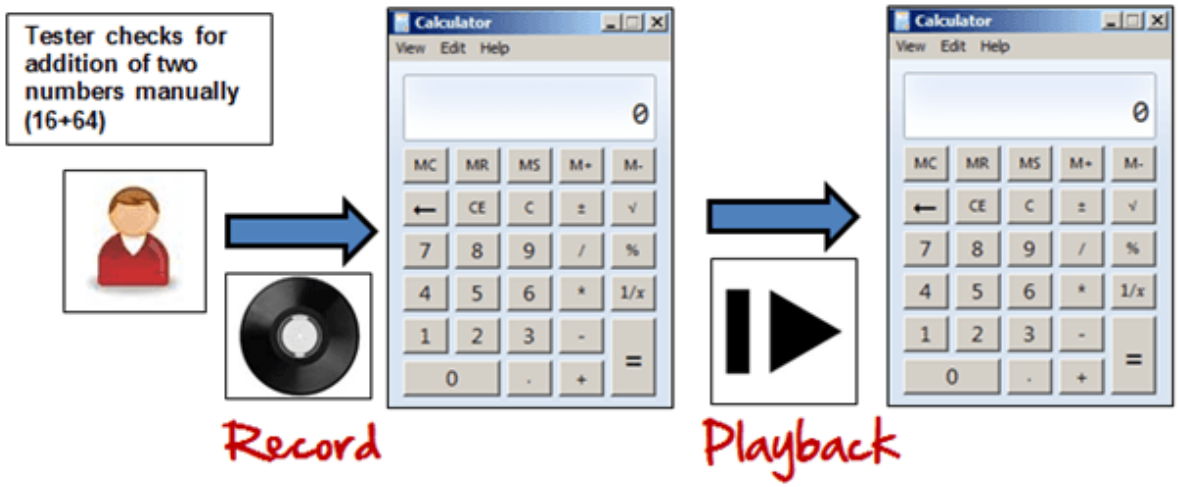


Figure 2 : Record and Play

1.2.4.3 Model Based Testing Process

Model here is a graphical description of system's behaviour which help us to understand and predict the behaviour of system. Models help in generating efficient test cases using the system requirements. Below points need to be considered for this model based testing:

Prepare and Build the model

Identify Inputs for the model

Calculate expected output for the model

Execute the tests

Compare the actual output with expected output

Decide further actions on the model

Modeling techniques from which test cases can be derived:

Charts - Depicts the state of a system, also checks the state after some inputs.

Decision Tables – These Tables used to determine results for each input applied

Model based testing is an evolving technique for the generation of test cases from the requirement specification. This method can determine undesirable states that your GUI can attain which is the main advantage of this method compared to above two methods..

Chapter Two: **Literature Review**

2.1 Mobile Operating Systems:

Operating system (OS) that allows smart phones, tablet PCs and other devices to run applications and programs is considered as Mobile Operating System. Mobile Operating System typically starts up when a mobile device gets power on, present screen with tiles and icons that present information and provide application accessibility. Mobile OS also handle cellular and wireless network connection functionality and phone accessibility. Some of mobile device operating system examples include Apple iOS, Google Android, BlackBerry OS, Symbian, webOS by Hewlett Packard (formerly Palm OS) and Windows Phone OS.

2.1.1 Android:

Google released the first Android OS by the name of 'Astro' on September 20th, 2008 (Develop Apps | Android Developers). After some period of time next upgraded versions 'Bender' and 'Cupcake' were released. Since then Google adopted the trend of naming conventions for Android as any dessert or sweet in alphabetical order. Other successive releases were Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich and Jelly Bean. Latest Version of Android is Marshmallow which is Android 6.0. Android platform is not closed like iOS by Apple, there are many good Android apps built by developers.

Android gained immense popularity into the smart phone and tablets market because of its beautiful appearance and efficient functionality. There were many new features introduced which played a important role in success of Android OS. Android is one of the top operating systems during current time.

2.1.2 iOS:

Apple introduced iOS was on 29th June 2007 when the first Apple's iPhone was announced (iOS - Apple (IN)). Post which iOS has been gone with many upgrades and the latest one is the iOS 9 version.

Apple has still kept iOS as closed Operating System and not allowed any other company to lay hands on its OS. Unlike Android, Apple has much concentrated on the performance along with user interface which the reason that the basic appearance of iOS is still same as it was in its first iOS version. Overall interface is user-friendly and is one of the mobile top operating systems in the world. Till now iOS has been used in all iPhones, iPod & iPad.

2.1.3 Symbian:

Symbian Operating System is developed by Nokia (All About Symbian). If any other company want to use Symbian then that company will have to take permission from Nokia before using it. Nokia has been a giant in the low-end mobile device segment, so except Java Symbian was the most used in mobile phones couple of years ago in the market. Till now Symbian is widely used in low-end devices but the demand for such devices has been continuously reducing. Nokia has upgraded Symbian mobile OS to made it capable to run efficiently on smart phones. ANNA and BELLE are the two latest updates of Symbian that are currently used in smart phones of Nokia. Unfortunately, Symbian Operating System is going downwards nowadays due to increasing popularity of Android and iOS Operating System.

2.1.4 Blackberry OS:

Blackberry Operating System is originally the property of RIM (Research In Motion) and it was first released in 1999 (THE MOBILE INDIAN). Research In Motion has developed this operating system for its Blackberry range of smart phone devices. Blackberry is much more different from other operating systems introduced. Interface style, and the Smartphone design, is different having a trackball on device for moving on the menu items and a qwerty keyboard for inputs.

Like Apple's iOS and Nokia's Symbian, Blackberry OS is also a close source Operating System which means it is not available for any other company. Latest release of this operating system is Blackberry OS 7.1. This version was introduced in May 2011

and it is being used in Blackberry Bold 9930 device. It is considered as a very reliable Operating System and is capable to resist to almost all the viruses or malware.

Some of the example of smart phones running on Blackberry OS are Blackberry Bold, Curve, Torch and Blackberry 8520 devices.

2.1.5 Tizen:

Samsung has co-worked with Intel to develop new operating system for mobile devices which is named as Tizen OS. This is a Linux-based platform built from Nokia and Intel's ditched MeeGo. It is an open-source, similar to Android OS, means that manufacturers who choose to adopt it are free to use with the interface to make it as unique as they are. Tizen is a flexible operating system built from needs of all stakeholders of the mobile and connected devices environment, including device manufacturers, network operators, mobile application developers and independent software vendors. Tizen is developed by a developer's community, under open source governance, which is open to all members who wish to contribute.

Tizen operating system is available in multiple profiles to serve as per different industry requirements. Current Tizen profiles are Tizen in-vehicle infotainment(IVI), Tizen Mobile, TV, and Wearables. Additionally, as of Tizen 3.0, all profiles are built on a common, infrastructure called Tizen Common infra.

Using Tizen OS, device manufacturer can start with any of these profiles and can modify to serve their needs or use the Tizen Common base to develop profile to meet the memory, power and processing requirements of device and quickly bring it to the consumers.

To meet the needs of specific customer segments Mobile operators can work with device partners to customize the OS and user experience. For developers and ISVs, OS offers the power of native application development with support of unparalleled HTML5. This Operating System also offers the potential for application developers to extend their reach to new "smart devices" running on Tizen OS, including wearable devices, consumer electronics, cars and other home appliance devices.

2.1.6 BADA:

Samsung owns an operating system that is known as BADA Operating System. It was designed for mid-range and high-end smart phone segment. Bada is significantly user-friendly and efficient operating system, much similar to Android OS, but unfortunately Samsung did not use Bada OS on large scale for unknown reasons.

Latest version of Bada 2.0.5 was released on March 15th, 2012. Till now only three phones are there which operates on Bada OS. These smart phones are Samsung Wave, Samsung Wave 2 and Samsung Wave 3. Bada would have achieved much greater success if Samsung had promoted it.

2.1.7 Palm OS:

Palm OS was targeted to design to work on touch screen Graphical User Interface (PalmOS Operating System). Palm OS was developed by Palm Inc in 1996 for PDA(Personal Digital Assistance) devices. After some Years, it was upgraded and it was able to support smart phone devices. Unfortunately, this could not make any place on the market and is not being used in any of the latest top devices.

It has been almost nine years since we saw the latest update in 2007 for Palm OS . Palm Operating System was used by many manufactures including Lenovo, Legend Group, Janam, Kyocera and IBM earlier.

2.2 Test Automation Approach of App Developers

Smartphone applications have recently gained popularity. Millions of smart phone apps are available on various application stores which give users various options to choose, however, this also raises concern whether these apps were tested before they released for users. In this study, we are exploring to understand the test automation culture among application developers working on Android. Specifically, we want to emphasis on current state of testing of applications, tools and frameworks that are commonly used by developers, problems faced by developers. A brief survey paper was published for such study in 2015 (Pavneet Singh, Ferdian, Nachiappan, Thomas, &

David, 2015). In this chapter we will present study on various Existing Mobile Application Testing Tools.

Existing Mobile Application Testing Tools/Frameworks:

This Section presents several existing tools and approaches for the automated testing of mobile applications. A number of different currently available automated tools is described in “Test Automation Tools for Mobile Applications: A brief survey” (Hughes Systique Corporation, 2013). Some of them belong to the standard tools that are used as a basis for more complicated approaches.

Techniques used in most of the available tools are formed from the keyword name and a related parameters list. Generally keywords are directed to a specific Graphical User Interface object for defining the parameters of objects. For defining the target object, keywords use references that accept either object ID or text content of the targeted object. Hierarchical structure of Graphical User Interface also used in the references taken, by defining the parents of the object searched. Many a times, using the hierarchy can be the only way out to identify an object uniquely. Some of the tools are based on references being described here.

2.2.1 Robotium:

Renas Reda, an international authority within test automation developed Robotium Tool in 2010. Robotium was the leading testing framework for Android Application Testing during the time. Robotium has been developed by many years of dedicated development. This is supported by a highly active community of developers. Robotium is used by developers across the world, which includes many enterprises as well as thousands of application studios related to Boutique. Robotium is a strong technical foundation for any Android Testing company. Still, the inhanced Operating System and hardware fragmentation creates challenges significantly, both from a quality perspective and cost point of view. Robotium founders introduced Robotium Tech to address the issues which enables Robotium users to further utilize the magnificant power of Robotium framework. Robotium Recorder is the first commercial which is offered by

Robotium Tech. Recorder provide Robotium users with an ultimate powerful productivity framework. This enables Android developers and quality engineers to create true Robotium-grade test scenarios.

Robotium is a User Interface testing tool designed for Android OS. It is suitable framework for tests automation for different successive Android versions and sub-versions. Tests Script created by Robotium used to be written in Java language. Robotium framework libraries were written for unit testing purpose. Instrumentation is abstracted in Robotium, which enables the preparation of grey-box automated test cases for applications. Robotium can be used both for applications with the source code available and without code information. With a help of Robotium, it is possible to write function, system and acceptance test cases, to find current activities and views and to make decisions automatically (User Guide Android Studio- Robotium Tech).

2.2.2 SIG-Droid:

SIG-Droid was introduced in 2015 by Nariman Mirzaei, Hamid Bagheri, Riyadh Mahmood and Sam Malek. A research paper is published in IEEE as SIG-Droid: Automated System Input Generation for Android Applications in 2015 (Nariman, Hamid, Riyadh, & Sam, 2015). SIG-Droid is a test framework for system testing of Android applications which is backed with automated analysis of framework for extraction of application models and symbolic execution of test script guided by models for obtaining test inputs which ensures covering various reachable branches in the program flow. SIG-Droid uses two automatically extracted models: Interface Model and Behavioural Model.

Interface Model was used to find values of an application that can be received through its interfaces and are then exchanged with symbolic values which deal with constraints with the help of a symbolic execution engine program. Post this Behavior Model was used to drive the application for symbolic execution and generate event sequences.

SIG-Droid uses symbolic execution technique, a promising automated testing technique which can effectively deal with constraints used. SIGDroid leverages the knowledge of Android's specification to automatically extract two models (Interface

Model & Behavioural Model) from an application's source code. Models are used for the generation of event sequences that aimed to simulate actual behaviours from user perspective. Behaviour Model used to capture event driven behaviour of the application including the relationships among event generators and event handlers. Interface Model represents all input interfaces in the application and the widgets including buttons, input boxes, checkboxes etc.

2.2.3 PATS :

Parallel Graphical User Interface Test Framework for Android Applications was proposed in 2015 by Hsiang-Lin Wen, Chia-Hui Lin, Tzong-Han Hsieh, and Cheng-Zen Yang (Hsiang-Lin, Chia-Hui, Tzong-Han, & Cheng-Zen, 2015). Framework consists of two kinds of nodes for testing: one node is testing coordinator and second is a set of testing slaves. PATS framework dynamically analyzes Graphical User Interface components for generating test scenarios. Reverse engineering approach was used in GUI Ripper technique. GUI structure was dynamically crawled to create tree for GUI as the state model for Application. For avoiding infinite explored visited GUI states of application, GUI Ripper checks the state depth threshold and state equivalence. Approach provides a broad testing coverage by going through testing all possible event sequences of Application. The main difference is that PATS generates event sequences for short-term testing on the set of slave nodes. Test Cases then assembled by these short-term event sequences and schedules them for the slave nodes of the framework.

2.2.4 MonkeyRunner:

Android SDK provided a tool to execute Test in Android devices called Monkeyrunner. Monkeyrunner provides an API for using commands to execute test scenario in Android emulators or Android devices (monkeyrunner | Android Developers).

APIs were used for writing programs which can control an Android device or emulator from outside. Using monkeyrunner, tester can write Python program to installs an Android application or testing package, executes it, sends key codes to it, captures screenshots of user interface, and saves screenshots on the workstation PC.

Monkeyrunner tool was initially designed to test applications and devices at the functional level or at framework level and for executing unit test suites for application under test, but tester are free to use it for other purposes of testing. Specific system included in Android SDK is Monkey, including Monkey tool and MonkeyRunner. Monkey tool is running directly on the mobile device and allows the generation of random events, like key presses or touching screens to discover the potential bugs by searching the known error patterns. MonkeyRunner is an API build on Monkey tool that enables functional testing and requires writing Python scripts to manage the testing process. It allows the sending of key events, taking screenshots of GUI and programmatically controlling the testing process on multiple Android devices at the same time. MonkeyRunner can compare screenshots with reference images to validate the visual correctness of the GUI.

2.2.5 MobiGUITAR:

This framework was introduced to support a wide variety of model-based Graphical User Interface testing techniques. The innovation is in the architecture of GUITAR framework that uses plug-ins to support extensibility and flexibility. MobiGUITAR uses three basic steps: ripping, generation, and execution.

Mobi-GUITAR acquired a state machine model of Graphical User Interface and it uses algorithms which better suited for mobile Application Platforms (Domenico, Anna, & Porfirio, 2015). Ripper in Mobiguitar is an enhanced version of Android Ripper. It first launched the app in a given start state and then obtain events list which can be performed on the Graphical User Interface in the state. This list is added with each event as a separate task in a task list which is used to fire events. Ripper removes an element from the task list and then fires that. As an outcome new states occur and the GUI's focus changed as the events fired. At the change in current state the ripper obtains the list of new events that can be fired and appends to the task list in a way that the path from the start state is pretended to each event. Thus formally, task is a sequence of events that always begins with an event in the start state that can be fired.

2.2.6 ReWeb and TestWeb:

ReWeb and TestWeb was initially developed for testing of Web Application and also it supports analysis of Web applications (Maryam, Rosziati, & Noraini, 2015). ReWeb accepts pages of a Web application to analyse for building its UML model in accordance with Meta model of the web page of the Application. TestWeb was used generates and execute test cases suit for a Web application whose model used to be generated by ReWeb technique. ReWeb and TestWeb were enhanced for testing mobile application for Android Operating System. This approach is to work as an add-on with Eclipse IDE such that it accepts java source code of application under test as and input. Proposed adaptation model includes two rudiments for testing web application as in the case of Reweb and TestWeb. Mobile_Analyzer is the first part that analyzes the source code of the software and derives the UML diagram while the other part generate test cases and executes testing scenarios. Mobile_Analyzer used to pass the input to the Apk_Analyzer. ApkAnalyzer is an open source application that helps to generate UML from android apk . Apk_Analyzer ias a virtual analysis tool, which is used to analyze API references, view application architecture and dependencies, and disassemble bytecodes in android applications. Apk_Analyser is a complete tool chain that supports modification of the applications with more printouts. In the proposed model, Apk_Analyzer accepts the java script source code as input and display its UML use case diagram as a output. UML diagrams are then passed to the test generator to generate the test cases for the Application under Test. Test cases used to be selected based on the test criterion or specification and passed to the test case executor module. For minimizing the generated test cases and improve the completeness, use case specification used to be based on functional and scenario. Use case specification used in the model used to be based on the following: (a) Identifying the functions of the SUT. (b) how to determine if SUT is properly working (c) Testing every functionality based on scenario test cases, one by one and (iv) Validate if the SUT result.

2.2.7 Appium:

Appium has been developed for automation testing of native, mobile , and hybrid applications (Appium: Mobile App Automation Made Awesome.). It is available as an open source Tool. Web apps which used to access using a mobile browser are called as Mobile Web Apps. Hybrid apps have native controls that enable interaction with web content using wrapper around a web view. Appium uses a webserver that exposes REST API. Web server receives connections from client, listens for commands, transmit those commands on a mobile device for execution, and responds with an response as a HTTP response which represents the result of the executed command . Concept of client/server architecture provide lot of possibilities like; we can write test scripts in any language which has http client API, but it is much easier to use libraries of Appium client. We can put the server on different machine and our tests are running on different. Test code can be written and rely on a cloud service to receive and interpret the commands. Test Script Execution is always performed in the context of a session. Initially Clients create a session with server in specific way to each library but they finishes by sending a POST request to the server, with a JSON object. Now server will start up the session for Automation with responding a session ID used for sending commands further. Jason Object" Desired capabilities" are set of keys and values sent to the Appium server which tell the server about the kind of automation session interested. There are also various capabilities which can be used to modify the behaviour of the server during automation execution. As an example, we might set the platformName capability to iOS to intimate Appium that Tester want a session for iOS , not for a Android device. Also we might set the safariAllowPopups capability to ensure that a Safari automation session has been allowed to use JavaScript for opening up new windows.

2.2.8 Image Template Matching :

An automated software testing system invented by John A. Gregory et al. (M. Pope, F. Stone, & John, 1994)can be used to automate the testing and to compare the design of different versions of the system. During the execution of the system under test,

the tool records all inputs, such as keystrokes and mouse events, and saves them in the script. Furthermore, it captures the screen images of the system. When the next version of the system under test is executed, the tool plays the inputs recorded before to operate the system and captures the screen images again. The screenshots of the first version of the system are compared to the correlating screenshots of the second version called by the same inputs. The tool displays the differences of the images and shows what components of the UI design have been changed. This tool does not evaluate the final design of the system, but demonstrates the visual differences between two versions of the same screen via image capturing. This technique could be used to compare the real application screen with the UI image created by the designer.

2.2.9 Sikuli:

Sikuli is developed by Tom Yeh, Tsung-Hsiang Chang and Robert C. Miller. Sikuli image recognition tool to automate the testing experience of the GUI which includes visual scripting API with an integrated development environment (Michael, Nikolaus, & Tom, 2014). Sikuli is based on the finding of target patterns on the screen and does not need access to the source code. Therefore it can be used both for desktop and mobile applications. However, mobile applications can be tested only on the desktop screen running in simulator or getting the application screen on the desktop connecting the device via VNC server. The second option can be used for Android applications without problems, but not for iOS devices, since VNC servers are not available for iPhone/iPad. The tool provides two core functionalities – Sikuli Script and Sikuli Search. With Sikuli Script it is possible to write visual scripts in Jython (combination of Java and Python) and to refer to UI elements using the provided library of functions and action commands. It allows the taking of a screenshot of the needed GUI component, adding it to the script and defining the action that this element should perform. The tool searches for a given component on the applications screen with a pattern matching technique, using opensource computer vision library. It compares the target pattern to each region on the screen of the same size, trying to find the most similar one, and is suitable for small patterns, such as buttons or icons. Sikuli also has an algorithm to detect larger patterns,

like a window or dialog box using a combination of matching elements in the relation to the target pattern. Applying grayscale or multiple scales to small elements, Sikuli is able to identify color change and resized images to detect possible changes in screen resolution. The system also provides the possibility to find text elements using optical character recognition (OCR). Sikuli Search is a part of the system that enables the search of information about the selected UI object in the online documentation. It contains mentioned three components: First is a screenshot engine, Second is a UI for querying the search engine and third is a UI for adding screenshots.

Framework / Features	Functional Classification	System Testing	Application Testing	Testing Multiple Devices Parllely	Devices Grouping Concept	Generic script based (Hierchical Libraries)	Tuple Concept for Easy Maintability and Reusability	Automatic Analysis (Output interpritation & bug Detection)
SIG-Droid (IEEE, 2015)	Behaviour and Interface Driven	Yes	Yes	No	No	No	No	Less
Mobi-Guitar (IEEE,2015)	GUI ripping, generation, and execution	Yes	Yes	No	No	No	No	Less
PATS (IEEE, 2015)	Parlel GUI Ripping and Execution	Yes	Yes	Yes	No	No	No	Less
ReWeb & TestWeb (IEEE, 2015)	UML Based Application Testing	No	Yes	No	No	No	No	Less
Robotium (Apache License, 2014)	For application Testing. (Create Event Based TCs)	No	Yes	No	No	No	No	Less
Sikuli (IEEE, 2014)	Image recognition to control GUI components	No	Yes	No	No	No	No	Less
Appium (appium.io)	Native Apps and System Testing with background API Support	Yes	Yes	Yes	No	Partial	No	Less
Proposed Framework	**Generic with Background API Support (Hierchical Libraries)**	Yes	Yes	Yes	Yes	Yes (Most)	Yes	Most

Figure 3 : Comparison of Available Framework

2.3 Programming languages for mobile testing:

In recent years, Mobile application development industry had been reached at the stage which was unexpected for most of us. It is changing the way of businesses function across the world. Enterprises are aligning mobile applications to their productivity with the rapid innovation in mobile devices across platforms. Enterprise calls for mobile app developers to write versions of an application for many different platforms with a common language with of reusable code. To realize the idea, it's time to validate, and narrow down on the platform you would like to build your application. As soon as it is decided, it’s time to select language, keeping in mind the strategy which you build.

2.3.1 Python:

Python was developed through the PEP (Python Enhancement Proposal) convention. PEP process was the basic mechanism to propose major new features, and to collect input on an issue from community, and for documenting the design decisions which have gone into Python development. PEPs were reviewed and commented by the Python community (Our Documentation | Python.org). The major academic conference on Python was named as PyCon. Many special mentoring programmes like the Pyladies were there.

Python is high-level language which is widely used, general-purpose, and interpreted dynamic programming language. Design philosophy of Python emphasizes on code readability, syntax that allows programmers to express concepts in few lines of code than in languages like Java or C++. Language provides constructs which is intended to enable clear programs on small and large both scales.

Python language supports many programming paradigms which includes object-oriented, functional programming or procedural style programming. It provide a dynamic type system with an automatic memory management and has a large comprehensive standard libraries. Python interpreters are available for variety of operating systems that allow Python code execution on different systems. By using third-party tools, like Py2exe, Pyinstaller, code can be packaged into various executable programs for some of

the popular operating systems by allowing the distribution of Python-based software to use on vartous.

CPython is the implementation of Python which is free and open-source software; also follows development model based on community based model for it for all of its development. It is managed by the Python Software Foundation which is non profitable organization. Most of the Python implementations can function as a command line interpreter, by which the user writes sequential statements and receives the immediate results.

Python integrated development environments are also consist browser-based IDEs, intended for science and math-related development and hosting environment.
Language Enhancement goes along with CPython development. Public releases of Python come in three categories. Version number used to increment considering these distinct types.

Major releases are those which are largely compatible and introducing new features in the version. In this case the second part of the version number used to increment. Such releases used to schedule to occur every second year and each major version supported for bug fixing after its release for several years. Bugfix releases do not include any new features but only the fixing of bugs reported. In this case, the third and final part of the version number used to increment for the version number update. Such releases are made periodically when sufficient number of bugs have been fixed since last released version. Security vulnerabilities also used to fixed in such releases versions.

2.3.2 Java

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995 which was later acquired by the Oracle Corporation. It provided a system for developing application software and deploying it in a cross-platform computing environment.Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for Microsoft Windows (still works

for XP, while only later versions currently "publicly" supported), Mac OS X, Linux and Solaris.

The Oracle implementation is packaged into two different distributions: The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end users, and the Java Development Kit (JDK), which is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.

2.3.2.1 Java Platform

Portability was one of the design goal of java, that means programs designed for the Java platform may run same as any combination of hardware and operating system with an accurate support of runtime. By compiling the java language code this can be achieved to an intermediate depiction called Java byte code, rather than directly to architecture particular machine code. Java byte code are intended to be executed by a virtual machine which is particularly written for the host hardware, but it is considered that Java byte code instructions are comparable to machine code. The End users use a Java Runtime Environment (JRE) which is installed either on their own machine for standalone Java applications or for java applets in a web browser.

Graphics, networking and threading are some of the host specific features which were provided to access in a generic way by Standard libraries. The porting can be simplified by the use of universal byte code. Although, the overhead of interpretation of byte code into machine instructions makes programs always run more slowly than native executables. Just-in-time was introduced for compilation of byte codes to machine code during runtime. Java is a platform-independent language which means it can be adapted to any platform to run on by a Java virtual machine (Java Software | Oracle).

2.3.2.2 Uses

Java was choosen to be used as a key pillar by Google and Android Inc. In the creation of the Android operating system which is an open source mobile operating system. Whilst the Android operating system which was built on the Linux kernel was

written mostly in C,the java language was used by the Android SDK as the basis for Android applications. Though, Android using Java byte code as an intermediate step while not using java virtual machine and finally targeting Android's own Dalvik virtual machine or more recently Android Runtime which actually compiles applications to native machine code upon installation.

Android also does not provide the full Java SE standard library, whilst the Android class hierarchy has an independent subset for its implementation.

2.4 Literature Gaps:

Current work done in the field of mobile application testing shows that there are still holes in effective automated testing considering the visual representation of the GUI elements. In particular, currently available Mobile Application Testing framework has various limitations and seems to be especially hard for mobile companies to adapt for their Native Applications Testing. In recent years there has been an increased focus in this area.

Prior Studies have generally focused on the way to establish a medium between Automation Tool and the lower level of mobile devices in order to execute the scenarios automatically. A well defined approach to adapt such tools have rarely focused

Scripting efforts are rarely considered. Reusability of test scripts is the area where we require a framework with well defined hierarchy so that scripting efforts can be optimized.

Approach for making generic scripts for different devices with similar functionality has not been addressed which should be addressed in framework which would give an edge to mobile testers

Making a group of device dynamically as per requirement for particular scenario is not addressed. Devices grouping as Master and Support devices as per test case requirement is an important area which should be implemented at framework core.

Maintainability is the important area for reusing the scripts for long time with the UI changes which should be focused at framework level.

Chapter Three: **Proposed Work**

Current work done in the field of mobile application testing shows that there are still holes in effective automated testing of UI design, considering the visual representation of the GUI elements. In this chapter a hierarchal framework have been proposed to address the limitation with the existing framework in the literature review.

3.1 Data Driven Model:

Main principle which guided the project implementation were to implement the variability to elements such as environment, end points, test data, handling of test devices, core protocols, locations separated out from the test logic (scripts). We introduced library approach. All test paths has been divided into functional parts and put in methods. If such method could be divided into other smaller functional parts which could be used in other methods – new methods could be made. All methods have been grouped into thematic libraries resulting in creation of hierarchical Framework. Objective representation of the mobile UI is accomplished by DroidManager; however framework is not dependent only on that. For achieving the Test Scripts versatility and flexibility, This Framework was created. Moreover, for achieving utility, efficiency and ease of use .ADB(Android Debug Bridge) is a adaptable command line tool that makes you communicate with an emulator instance or connected Android-powered device. ADB lets you communicate and form a bridge for establishing communication between mobile device and PC running the tool. Core Methods are gathered in libraries at lower level. DroidManager contains methods which are used by all helper libraries. They are the lowest in terms of level of abstraction. Most of them are giving the basic UI interaction functionalities like pressing the buttons, clicking particular objects or sliding on the touchscreen. They are binding Droid engine and higher level methods.

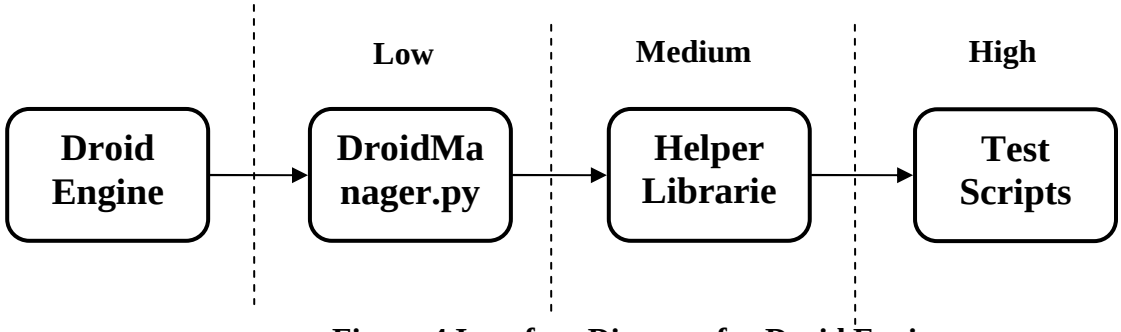


Figure 4 Interface Diagram for Droid Engine

Aside from above libraries which are implementing test cases execution paths, there are others, responsible for background functionalities.

One of the most important is *RunInThreads.py* which is taking care of connected samples management depending on the test script type. Different test cases need different samples count to be properly performed. For example sending SMS needs at least 2 samples; one is sending the message and the second one is receiving it. Call forwarding on the other hand needs at least 3 samples; one is making the voice call, second one is forwarding it, and at the end the third one is receiving the call. To handle all connected devices *RunInThreads* is setting them into groups which contain sample count needed to execute the test. Each group consists of 1 Master device and Slave devices. Masters and Slaves roles should be defined in GUI before the test. As it was explained before, core functionalities like sending messages, needed for performing test cases are grouped into thematic libraries like *helpers*. Test execution paths on the other hand are the highest level of Framework abstraction and are located in Test Scripts. Test Scripts consists mainly of test case logic and test path. They are only references to other Framework methods arranged respectively to test case characteristic.

Framework Architecture

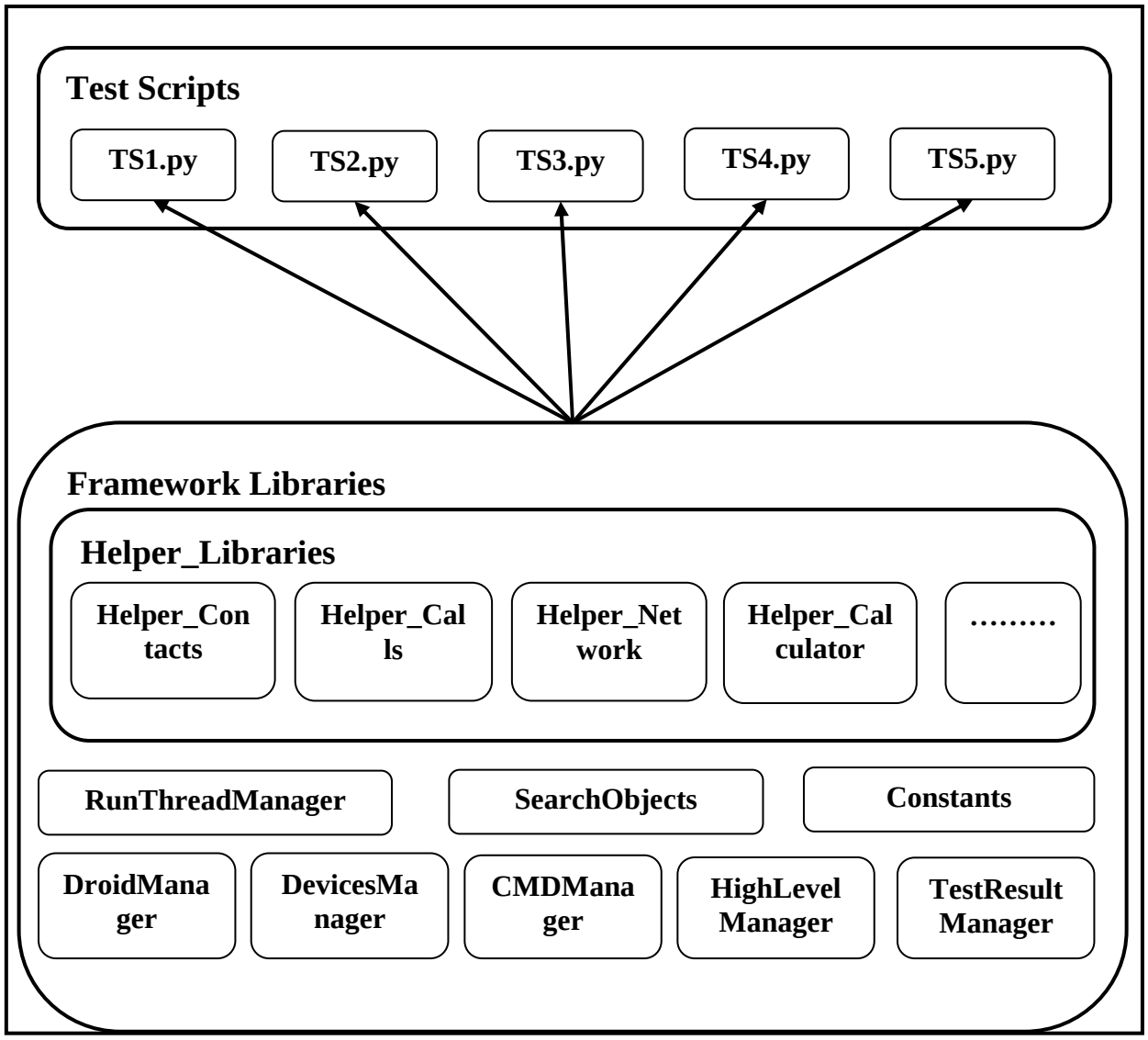


Figure 5 Framework Architecture

There are several ways to control a mobile device from a PC. Tool is using two of them:

ADB: Android Debug Bridge is a command line utility included with Google's Android SDK that allows communicating with connected Android devices. With ADB it is possible to execute Linux like powerful system commands and get access to lower layer functions like installing applications, running activities & intents, pulling & pushing files, getting device properties.

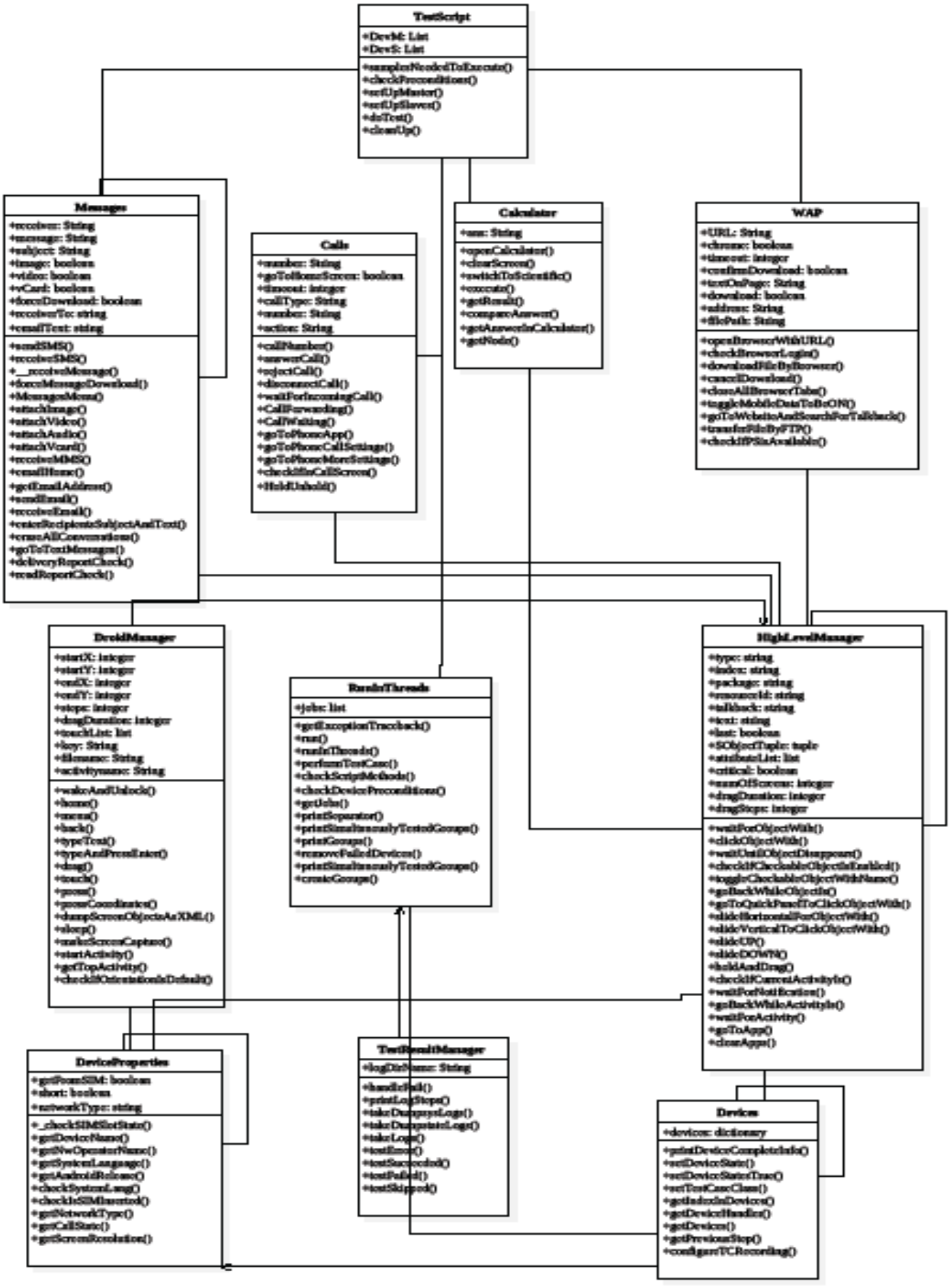
Droid Engine Libraries: Droid Engine interact with the device and use intents, UI Objects, Activities to interact with device. ADB and Platform Droid Engine are therefore responsible for communication and controlling the mobile devices from PC.

3.2 Class Diagram for the Hierarchy

Helper libraries are the classes in which functions are grouped applications wise. These functions uses lower level libraries like HighLevelManager, DroidManager in the functions. DevicesManager and RunThreadManager controls execution of test scripts.

As it was explained before, core functionalities like sending messages, needed for performing test cases are grouped into thematic libraries like *helpers*. Test execution paths on the other hand are the highest level of Framework abstraction and are located in Test Scripts. Test Scripts consists mainly of test case logic and test path. They are only references to other Framework methods arranged respectively to test case characteristic.

Model::Main


TestScript
+DevM: List
+DevS: List
+numplesNeededToExecute()
+checkPreconditions()
+setUpMaster()
+setUpSlaves()
+doTest()
+cleanUp()
Messages
+receiver: String
+message: String
+subject: String
+image: boolean
+video: boolean
+vCard: boolean
+forceDownload: boolean
+receiverTo: string
+emailText: string
+sendSMS()
+receiveSMS()
+__receiveMessage()
+forceMessageDownload()
+MessagesMenu()
+attachImage()
+attachVideo()
+attachAudio()
+attachVcard()
+receiveMMS()
+emailHome()
+getEmailAddress()
+sendEmail()
+receiveEmail()
+enterRecipientsSubjectAndText()
+cancelAllConversations()
+goToTextMessages()
+deliveryReportCheck()
+readReportCheck()
Calls
+number: String
+goToHomeScreen: boolean
+timeout: integer
+callType: String
+number: String
+action: String
+callNumber()
+answerCall()
+rejectCall()
+disconnectCall()
+waitForIncomingCall()
+CallForwarding()
+CallWaiting()
+goToPhoneApp()
+goToPhoneCallSettings()
+goToPhoneMoreSettings()
+checkIfInCallScreen()
+HoldUnhold()
Calculator
+ans: String
+openCalculator()
+clearScreen()
+switchToScientific()
+execute()
+getResult()
+compareAnswer()
+getAnswerInCalculator()
+getNode()
WAP
+URL: String
+chrome: boolean
+timeout: integer
+confirmDownload: boolean
+textOnPage: String
+download: boolean
+address: String
+filePath: String
+openBrowserWithURL()
+checkBrowserLogin()
+downloadFileByBrowser()
+cancelDownload()
+closeAllBrowserTabs()
+toggleMobileDataToBeON()
+goToWebsiteAndSearchForTalkback()
+transferFileByFTP()
+checkIfPSIsAvailable()
DroidManager
+startX: integer
+startY: integer
+endX: integer
+endY: integer
+steps: integer
+dragDuration: integer
+touchList: list
+key: String
+filename: String
+activityname: String
+wakeAndUnlock()
+home()
+menu()
+back()
+typeText()
+typeAndPressEnter()
+drag()
+touch()
+press()
+pressCoordinates()
+dumpScreenObjectsAsXML()
+sleep()
+takeScreenCapture()
+startActivity()
+getTopActivity()
+checkIfOrientationIsDefault()
RunInThreads
+jobs: list
+getExceptionTraceback()
+run()
+runInThreads()
+performTestCase()
+checkScriptMethods()
+checkDevicePreconditions()
+getJobs()
+printSeparator()
+printSimultaneouslyTestedGroups()
+printGroups()
+removeFailedDevices()
+printSimultaneouslyTestedGroups()
+createGroups()
HighLevelManager
+type: string
+index: string
+package: string
+resourceId: string
+talkback: string
+text: string
+last: boolean
+SObjectTuple: tuple
+attributeList: list
+critical: boolean
+numOfScreens: integer
+dragDuration: integer
+dragSteps: integer
+waitForObjectWith()
+clickObjectWith()
+waitUntilObjectDisappears()
+checkIfCheckableObjectIsEnabled()
+toggleCheckableObjectWithName()
+goBackWhileObjects()
+goToQuickPanelToClickObjectWith()
+slideHorizontalForObjectWith()
+slideVerticalToClickObjectWith()
+slideUP()
+slideDOWN()
+holdAndDrag()
+checkIfCurrentActivityIs()
+waitForNotification()
+goBackWhileActivityIs()
+waitForActivity()
+goToApp()
+closeApps()
DeviceProperties
+getFromSIM: boolean
+short: boolean
+networkType: string
+_checkSIMSlotState()
+getDeviceName()
+getNwOperatorName()
+getSystemLanguage()
+getAndroidRelease()
+checkSystemLang()
+checkIsSIMInserted()
+getNetworkType()
+getCallState()
+getScreenResolution()
TestResultManager
+logDirName: String
+handleFail()
+printLogStops()
+takeDumpsysLogs()
+takeDumpstateLogs()
+takeLogs()
+testError()
+testSucceeded()
+testFailed()
+testSkipped()
Devices
+devices: dictionary
+printDevicesCompleteInfo()
+setDeviceState()
+setDeviceStatesTrue()
+setTestCaseClass()
+getIndexInDevices()
+getDeviceHandles()
+getDevices()
+getPreviousStep()
+configureTCRecording()

3.3 Devices Grouping: Master/Slave Concept

Framework can handle multiple groups of test devices at same time depending on Test Script Type. Different test cases may need different count of Test Devices to be properly performed. For example sending SMS needs at least 2 samples – one is sending the message and the second one is receiving it. Call forwarding on the other hand needs at least 3 samples – one is making the voice call, second one is forwarding it, and at the end the third one is receiving the call. Tester may also want to perform same test case parllely on different Mobile Phone which can be overcome with the concept introduced.

Master devices are the devices which should be tested and return results. Slave devices are not to be tested, but could be used to help in testing. Master devices, i.e. when Test Script requires minimum two devices and we are testing only one Master.Each Test Script is concentrating on testing one Master device at a time with the help of Slave devices. Other Master devices are tested in parallel if proper Slave count is available.

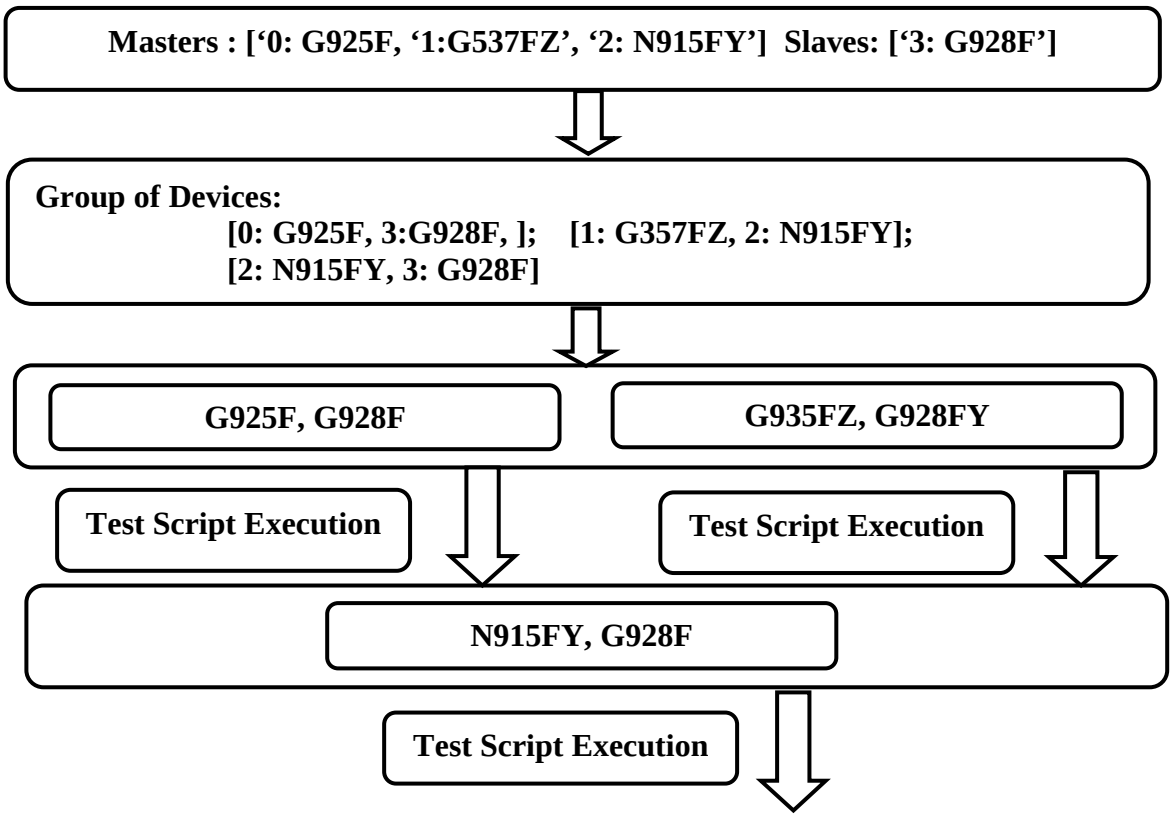
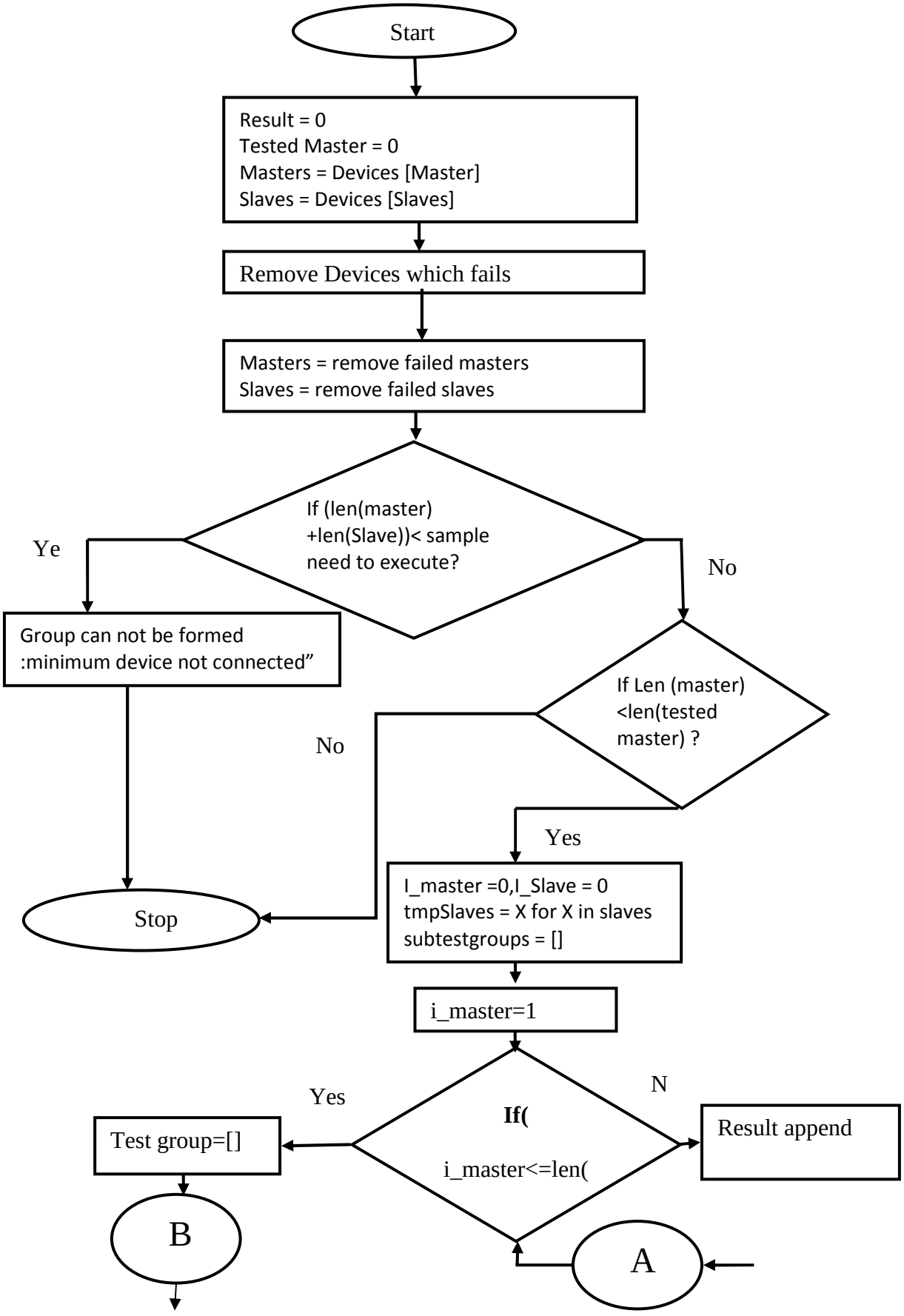


Figure 6 Devices Grouping Example with Parllel Execution


Start
Result = 0
Tested Master = 0
Masters = Devices [Master]
Slaves = Devices [Slaves]
Remove Devices which fails
Masters = remove failed masters
Slaves = remove failed slaves
If (len(master)
+len(Slave))< sample
need to execute?
Ye
No
Group can not be formed
:minimum device not connected”
If Len (master)
<len(tested
master) ?
No
Yes
Stop
I_master =0,I_Slave = 0
tmpSlaves = X for X in slaves
subtestgroups = []
i_master=1
Yes
If(
i_master<=len(
N
Test group=[]
Result append
B
A

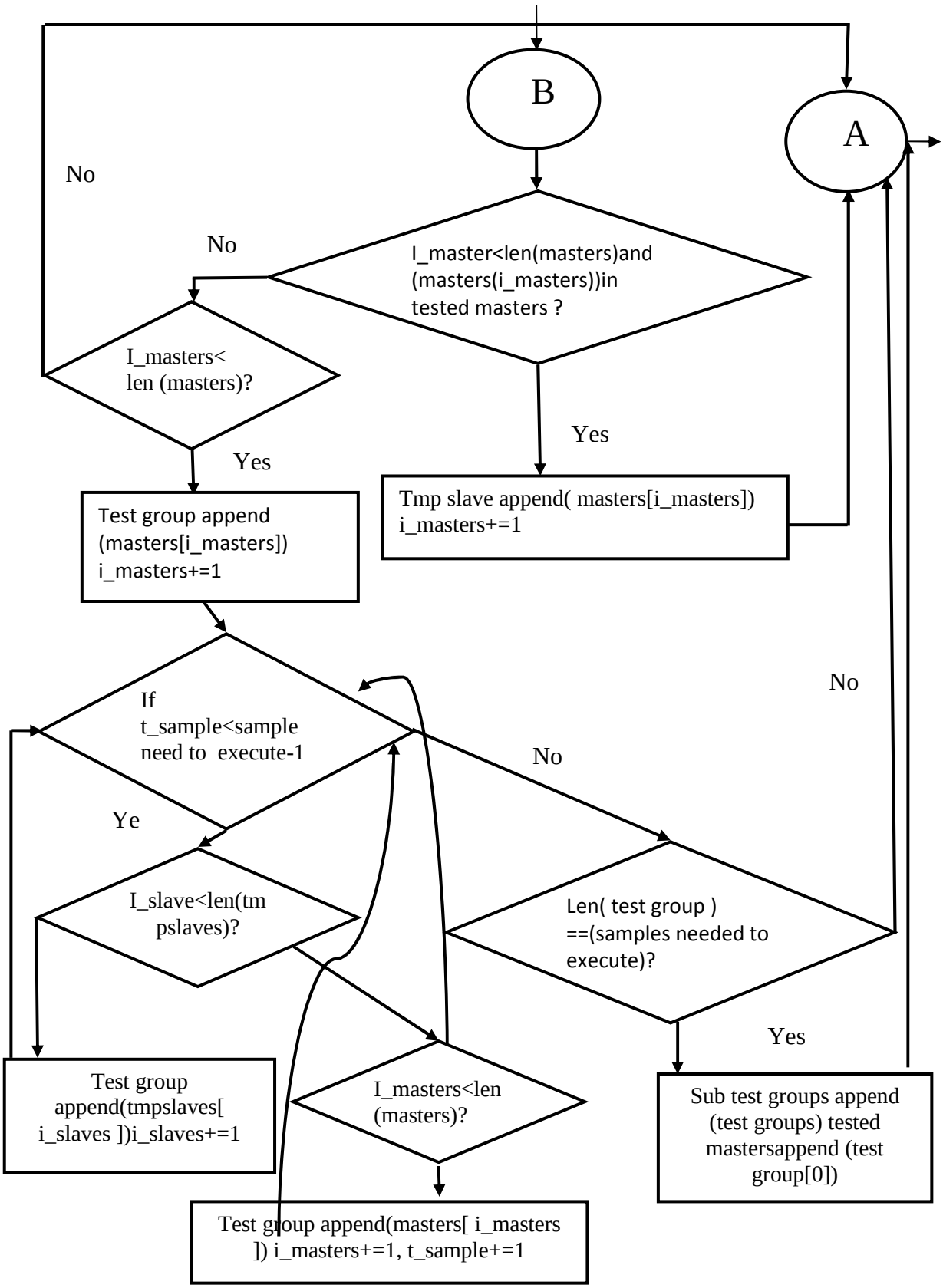


Figure 7 Flow Chart for Grouping Algorithm

3.4 Scripting Approach:

Android provides methods and mechanisms allowing certain UI functionalities to be performed with UI interaction at its top layer and without UI interaction at its lower layers using Intents, KeyCode. All intent, attributes, object information are captured using Dumpsys for scripting.

3.4.1 Raw Scripting Approach

Tester can for simulate simple methods like pressing hardware buttons ,touch objects, clicking objects on the mobile screen. Droid Engine provides methods like press(), or clickObject() with which interaction with mobile device User Interface is possible. Such methods can be used to write test scripts with desired test scenarios.

TS1.py

```
……
press(“KEYCODE_HOME”,DOWN)
press(“KEYCODE_HOME”,UP)
setSearchObject(text = “Applications”)
clickObject(u“Application”, u“Applications”, “android.widget.TextView”, “”,0)
setSearchObject(u“Messages”, u“Messages”, “android.widget.TextView”, “”,0)
clickObject(u“Messages”, u“Messages”, “android.widget.TextView”, “”,0)
…….
```

TS2.py

```
……
press(“KEYCODE_HOME”,DOWN)
press(“KEYCODE_HOME”,UP)
setSearchObject(text = “Applications”)
clickObject(u“Application”, u“Applications”, “android.widget.TextView”, “”,0)
setSearchObject(u“Messages”, u“Messages”, “android.widget.TextView”, “”,0)
clickObject(u“Messages”, u“Messages”, “android.widget.TextView”, “”,0)
…….....
```

Figure 8 Raw Scripting Approach Example

In above example there are two test scripts which are implementing some different test scenarios, but they are sharing at some point, same path. They are all opening *Messages* app by going to home screen, clicking *Applications* and then *Messages* object. Such approach is sufficient when we assume that test scripts are written only for specific device and the test path will not change. But if we would like to apply those test scripts to other device with different UI, we would have to prepare new test scripts or modify old ones. For above example, if new device will have object with text *Apps* instead of *Applications*, we would have to change this text in every place in all test scripts which could be very time-consuming.

3.4.2 Scripting Approach with Proposed Framework

To prevent constant modification in all places every time UI changes in new devices, library approach has been used. All test paths has been divided into functional parts and put in methods. If such method could be divided into other smaller functional parts which could be used in other methods – new methods was made. All methods have been grouped into thematic libraries resulting in creation of hierarchical Framework Examples of such thematic libraries are helpers which are grouping methods module wise. Example :

Lib/Helpers/messages.py : Methods related with messaging test cases: sendSMS(), receiveSMS(), sendMMS(), receiveMMS(), eraseAllConversation(), MessagesMenu()…

Lib/Helpers/calls.py : Methods related with calling test cases: callNumber(), answerCall(), disconnectCall(), rejectCall(), goToPhoneMoreSettings(), sendSSCode()…

Lib/constants.py : Storing most frequently used constants like delays, Packages Names

DroidPropertiesFileManager.py : Managing user configuration file which contains properties needed for Test Scripts and Framework, e.g. application accounts credentials

DeviceProperties.py : Getting the mobile device properties, e.g. network name, model name, network type, packet data availability, SIM state, etc.

Aside from above libraries which are implementing test cases execution paths, there are others, responsible for background functionalities. One of the most important is RunInThreads.py which is taking care of connected samples management depending on the test script type.

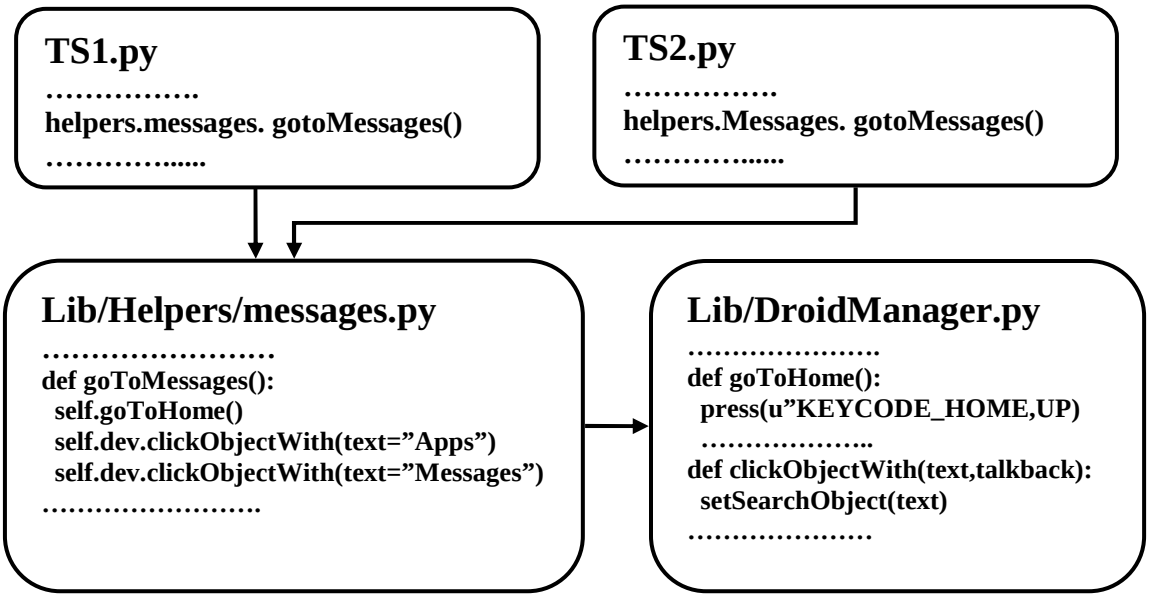


Figure 9 Test Script Writing Approach with the Framework

Above figure is showing implementation from figure given in raw scripting approach, but with the approach given. The code which was responsible for opening Messages app, was divided into smaller functional methods and was put in corresponding libraries. Now in test scripts instead of redundant code, we have only one line which is calling goToMessages() method, which is defined in Lib/Helpers/messages.py library. This compared to raw STP scripting gives us benefits such as:

Flexibility: No need to change all test scripts in case of small modification. Proper change can be done in corresponding method only and it will take effect automatically in all test scripts in which it is used

Versatility: No need to create different test script versions for different devices. All UI changes can be handled in corresponding methods which are maintained by Automation team to be compatible with all devices

Readability: Hierarchical framework structure makes test scripts easy to read as they only consist of logical references to high level methods which are defined in corresponding helper libraries

Saves development time: No need to write new Test Scripts from scratch, some functionalities could be already implemented

3.5 RunThreadsManager:

RunInThreadManager is the core for the execution of test script. All the classes which controls execution flow are defined in RunThreadManager. Device Grouping algorithm, Sequencing of Methods required to be followed, MultiThreading etc. all defined under RunThreadManager.

Perform Test Case Controller:

As it was explained before, core functionalities like sending messages, needed for performing test cases are grouped into thematic libraries like helpers. Test execution paths on the other hand are the highest level of Framework abstraction and are located in Test Scripts. Test Scripts consists mainly of test case logic and test path. Common Test Script structure has been made and divided into below parts.

Preconditions: checking if mobile device is well prepared for the particular test, e.g. correct sample count, SIM card inserted. Executed in parallel on all connected devices

Set Up: setting up the devices to meet the test requirements, e.g. changing network mode to 2G if testing some GSM only functionalities. There can be 2 types of Set Ups. Set Up Master: Executed on Master device from each test group; Set Up Slaves: executed in parallel on all Slave devices from each test group

Do Test: Implementing test case execution path – interactions b/w Master and Slaves

Result: TestResultManager used to take care of test result.Indicating proper end result: Pass: performed successfully and pass criteria were met

Fail: failed if pass criteria were not met or object was not found

Skip: was not performed due to not met preconditions

Error: encountered an exception due to Script or Framework exception

Clean Up: changing back settings which were modified during Set Up part. Executed in parallel on all devices from the test group

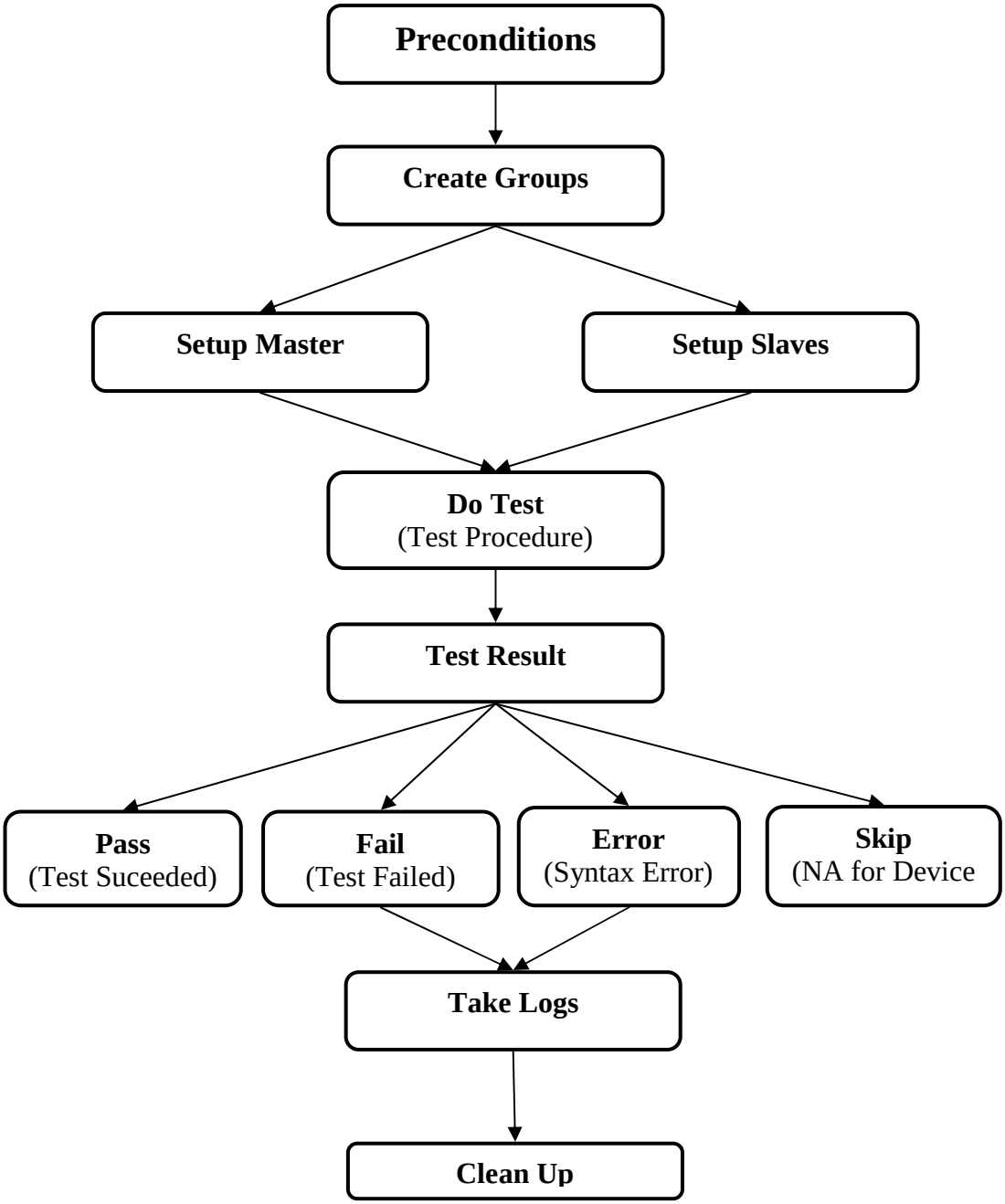


Figure 10 Test Script Execution Flow

Chapter Four: **RESULTS**

In this chapter we are evaluating the usability of the framework that has been presented. We will evaluate the areas which were discussed under literature gap in chapter 2.The first implementation task of the pilot is to create prototypes of the framework how well they have been met. it is that test automation is a very large and non-trivial subject.

4.1 Evaluating the Framework

Feature from the framework while working with the framework is grouping related test cases together into test suites. Libraries have been designed keeping functional cohesion and coupling focused. Libraries have been kept at different layers to group the similar functionality together.

Next tasks were writing helper classes to automated tests for Call and SMS functionality using both the data-driven and keyword-driven approaches and using them latter also in other modules which might need similar functionality in other modules. Libraries proceeded without any surprises and in the end automated tests could be executed for large scale. Libraries at core contain methods which can be used across all helper classes. Some of these methods are like below:

Return Type	Method and Description
Boolean	**typeText()** To type any text eg. in messages, emails etc
Boolean	**startActivity()** Start activities eg. Phone,Settings etc.
Boolean	**setSearchText/Talkback ()** Find object with text or talkback on the screen
Boolean	**checkIfCheckboxChecked()** Check if some options is enabled or disabled
Boolean	**clickCheckboxWithName()** Change status of options with checkbox
Boolean	**clickRadioButtonWithName()** Selection of radiobutton among the group
Boolean	**slideHorizontalToClickSObject ()** Slide horizontal to click something on the screen

Boolean	**home()/back()/menu()** Enters Home Screen/ Go to the previous screen/ Enters Menu
Boolean	**makeScreenCapture()** Captures Screenshot
Boolean	**unlockScreen()** Unlock Screen of the device
String	**getPhoneNumber()** Get own phone number from phonebook
String	**getScreenResolution()** Get Screen Resolution of the device
Boolean	**clickObjectWith()** Click Object on the screen with perticular attribute
Boolean	**hideKeyboard()** Hides the Key Board from the screen
Boolean	**scrollBackward()** Performs a backward scroll with the default number of scroll steps (55).
Boolean	**scrollBackward(int steps)** Performs a backward scroll.
Boolean	**scrollForward()** Performs a forward scroll with the default number of scroll steps (55).
Boolean	**scrollForward(int steps)** Performs a forward scroll.
Boolean	**scrollToBeginning(int maxSwipes)** Scrolls to the beginning of a scrollable layout element.
Boolean	**scrollToBeginning(int maxSwipes, int steps)** Scrolls to the beginning of a scrollable layout element.
Boolean	**scrollToEnd(int maxSwipes)** Scrolls to the end of a scrollable layout element.
Boolean	**scrollToEnd(int maxSwipes, int steps)** Scrolls to the end of a scrollable layout element.
Int	**getChildCount(UiSelector childPattern)** Counts child UI element instances matching the childPattern argument.
Int	**getMaxSearchSwipes()** Gets the maximum number of scrolls allowed when performing a scroll action in search of a child element.
Double	**getSwipeDeadZonePercentage()**

	Returns the percentage of a widget's size that's considered as a no-touch zone when swiping.

4.1.1 Degree of Coupling and Cohesion in the Framework

Higher Level of cohesion and lower level of coupling lead to good software design. Cohesion partitions functionality so that it is concise and closest to the data relevant to it, however decoupling ensures that the functional implementation is isolated from the rest of the system.

Low cohesion means that the class does a great variety of actions and is not focused on what it should do. High cohesion would then mean that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.

As explained in the class diagram given there are different libraries at different layer which have grouped the functionality category wise.

DroidManager - Contains all the methods which directly interact to with the Lower Layer of Android

HighLevelManager – Contains all the methods which are common and will be required for execution of various steps in different mobile applications

TestResultManager – All the functionality related to Test Results Management have been kept under this

Constants – All the constants which would be required in Test Scripts would be kept here

Devicemanager – Functions related to devices management are kept under this libraries

CLI Manager – All the functionality which needs to be controlled from CLI are covered in this library

Helper Classes – Different Helper Classes used to be written as per the Requirement. For Message Application Testing all the functions related to Messaging would be kept here. Whenever tester needs any function related to message, he/she can find in this libarary.

*D*ecoupling allows you to change the implementation without affecting other parts of your software. The most effective method of decreasing coupling via interface. As explained all the modules are separated and mostly independent So changes at lower level can be made without affecting the other parts. Framework at core hardly get changed as all the functions written either interact at Lower Level of Android or its related with device/framework management/Result Management. This way, higher degree of Cohesion and Lower Level Of Coupling have been achieved in the framework.

4.1.2 High Level Requirements

- The framework MUST execute test cases automatically. That includes also for example verifying results, handling errors and reporting results.
- The framework MUST be easy to use without programming skills.
- The framework MUST be easily maintainable.

4.1.3 Automatic Test Execution

- The framework introduced is able to execute tests unattended after starting the Test.
- Tester should be able to start and stop test execution manually at any time.
- It is possible to start test execution automatically at any time which might be predefined.
- After certain events tester should be possible to start test execution automatically.
- There should be handling for Non-fatal errors caused by the Device Under Test or the test environment must without aborting the test execution.
- Test results must be verified.
- Every executed test case must give output as either Pass, Fail or Script Error
- Error and Failed test cases should have a short message explaining the failure reason.
- Logs should be maintained for Test execution.

- Test execution SHOULD be logged using different, configurable logging levels.
- Result report must be created automatically after test.

4.1.4 Ease of Use

- The framework should use keyword-driven or Object driven approach.
- The framework must support creating Helper Classes to group similar functionality together.
- The framework should be able to support specifying common set up and tear down functionality for test cases.
- The framework should support grouping of test devices dynamically with the execution of test script into test suites.

4.1.5 Maintainability

- The framework must follow modular approach for easy maintainability.
- The framework should be implemented using object oriented programming languages.
- The testware in the framework must be under version control.
- The framework must have coding and naming conventions.
- Framework must have followed coding documentation standard. .
- Usability of the tested system must be increased as much as possible.

However, that the framework works well at least in similar contexts as in the pilot. Proving that it works in general in system and component level acceptance testing would require a larger study but there does not seem to be any problems preventing it. Based on the positive experiences and good results it can be asserted that the framework concept presented in this thesis is valid.

Chapter Five: **Conclusion and Future Work**

This report intended to present a new Data Driven Framework based on Python Language. Framework has concept of grouping related test cases together into test suites. Framework designed in Chapter 3 is based on the requirements gathered in Chapter 2. Using data driven and hierarchal approaches were considered most important lower level requirements in terms of ease-of-use and maintainability and the presented concept naturally had these capabilities. Although the experimental results are preliminary, yet in the pilot run everything went pretty much as expected. The approach worked very well and proved to be useful. Keyword-driven pilots proceeded without problems as well and results were even better than anticipated. Pilot experiences were collected together and based on them the overall framework was declared feasible. Changes affected also detailed requirements specified earlier and a revised requirement set was presented.

Framework derives many interesting issues which can be further investigated in the future implementation. As example, here are some crucial issues to the practical use of it. First, we have found the inconsistency problem for the Open GL Applications like camera, Browser etc. For Such Applications libraries can be developed based on OpenCV to access applications using Image Processing methods. I hope that in the near future the framework can also be released as an open source in one format or other. That way ideas presented in this thesis would be easily available in a functional format for everyone and other people could extend the framework into new directions. While waiting for that to happen, however, it is time to write the final period of this thesis.

Chapter Six: **References**

All About Symbian. (n.d.). Retrieved 05 15, 2016, from http://www.allaboutsymbian.com/: http://www.allaboutsymbian.com/

Appium: Mobile App Automation Made Awesome. (n.d.). Retrieved May 07, 2016, from http://appium.io/: http://appium.io/introduction.html?lang=en

Develop Apps | Android Developers. (n.d.). Retrieved 05 14, 2016, from Android Developers: https://developer.android.com/index.html

Domenico, A., Anna, F. R., & Porfirio, T. (2015). MobiGUITAR Automated Model-Based Testing of Mobile Apps. *Software* , 53-59.

Hsiang-Lin, W., Chia-Hui, L., Tzong-Han, H., & Cheng-Zen, Y. (2015). PATS: A Parallel GUI Testing Framework for Android Application. *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. Volume:2*, pp. 210-215. Taichung: IEEE.

Hughes Systique Corporation. (2013). Test Automation Tools for Mobile Applications: A brief survey. 11.

iOS - Apple (IN). (n.d.). Retrieved 05 13, 2016, from http://www.apple.com/ios/what-is/: http://www.apple.com/ios/what-is/

Java Software | Oracle. (n.d.). Retrieved 04 20, 2016, from Oracle Technology Network for Java Developers | Oracle Technology Network | Oracle: https://docs.oracle.com/javase/tutorial/

M. Pope, G., F. Stone, J., & John, A. G. (1994). *Patent No. US5335342 A.*

Maryam, A., Rosziati, I., & Noraini, I. (2015). Adaptation Model for Testing Android Application. *Computing Technology and Information Management (ICCTIM), 2015 Second International Conference* (pp. 130-133). Johor: IEEE.

Michael, K., Nikolaus, C., & Tom, Y. (2014). Abstracting Perception and Manipulation in End-User Robot Programming using Sikuli. *2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)* (pp. 1-6). Woburn, MA: IEEE.

monkeyrunner | Android Developers. (n.d.). Retrieved May 05, 2016, from http://developer.android.com: http://developer.android.com/tools/help/monkeyrunner_concepts.html

Nariman, M., Hamid, B., Riyadh, M., & Sam, S. M. (2015). SIG-Droid: Automated System Input Generation for Android Applications. *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on* (pp. 461-471). Gaithersbury, MD: IEEE.

Our Documentation | Python.org. (n.d.). Retrieved 04 24, 2016, from Python Software Foundation [US]: https://www.python.org/dev/peps/

PalmOS Operating System. (n.d.). Retrieved 05 14, 2015, from PalmOS Operating System: http://www.operating-system.org/betriebssystem/_english/bs-palmos.htm

Pavneet Singh, K., Ferdian, T., Nachiappan, N., Thomas, Z., & David, L. (2015). Understanding the Test Automation Culture of App Developers. *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)* (pp. 1-10). Graz: IEEE.

THE MOBILE INDIAN. (n.d.). Retrieved 05 15, 2016, from THE MOBILE INDIAN: http://www.themobileindian.com/talktime/question/What-is-the-operating-system-in-Blackberry-Q-

Tschernuth, M., Lettner, M., & Mayrhofer, R. (2011). *Evaluation of Descriptive User Interface Methodologies for Mobile Devices.* Berlin: Springer-Verlag.

User Guide Android Studio- Robotium Tech. (n.d.). Retrieved 05 10, 2015, from Robotium Tech: http://robotium.com/pages/user-guide-android-studio