Selection and Implementation of Test Framework
For Automated System Test of Mobile Application
Acknowledgements

I am forever thankful for the wisdom and guidance my advisor Mr Stoyan Todorov and Mr Andreas Fetzer has provided me over the past one year. They have spent an incredible number of hours together with me in defining and refining the necessary components of this thesis. Thanks to their continuous support and pushing I finally completed this task.

My deepest thanks goes to my friends who had shown patience for me not spending time with them and skipping weekends for work. Their love and support over the long years made it possible for me to finish this work.

Last but not least I would like to thank my parents, who, without questioning have always provided me the necessary support so that I was able to acquire the education I wanted.

Stuttgart, 23 January 2016

Ankit Shrivatri.
Abstract

Software Quality is a key concern for any companies working with software development. This is true due to the fact that the success of any software directly depends on Quality of software. It is expected that the software is of best quality for a long duration of time. With the introduction of Mobile applications the task of maintaining the quality of an application has been difficult and have faced many challenges.

Many companies working with mobile application have reformed their process in order to maintain the quality of their application. The introduction of Automation testing in the test process is one such reform that have changed the face of mobile application testing in today’s world.

This work deals with the concepts of Automation System testing for the mobile application which is until now a new thing and it has many things yet to be explored. The approach to automation testing is simple yet unique for the department of PT-MT/Quality Management in Robert Bosch GmbH based in Leinfelden, Stuttgart. Over here a selection and implementation of a test framework will be done for Automation testing of the mobile Applications that are being developed.

For this a requirement specification document is being created which will form the basis for selecting a framework from the KT Analysis table. Finally, a framework TestComplete will be implemented for the already developed application "PLR measure&go"

The implementation will include all the procedure required to set up the test framework as a part of documentation. The framework TestComplete will be used to create System test for iOS and Android operation system. Lastly the execution of test and the Result reporting is being shown as a complete process for Automation testing.

Key words: Software quality, software testing, automation testing, mobile application, TestComplete, android testing, ios testing
# Contents

Acknowledgements .......................... i
Abstract .................................. iii
List of figures ................................. ix
List of tables ............................... xi

1 Introduction ............................... 1
  1.1 Motivation ........................................ 1
  1.2 Problem Statement .............................. 2
  1.3 Aim of the Master Thesis ........................ 3
  1.4 Structure of Thesis .............................. 3

2 Fundamentals ............................... 5
  2.1 Software Testing .............................. 5
  2.2 Mobile Application in general .................. 7
  2.3 Types of Mobile Apps ............................ 8
  2.4 Software Development Process ................... 9
    2.4.1 Software Lifecycle .......................... 9
    2.4.2 Manual and Automation Test .................. 15
    2.4.3 Benefits of Automation Test .................. 18

3 State of Art ................................ 21
  3.1 Process of Mobile testing in a Project ........... 21
  3.2 Effort Estimation for Manual and Automation Test 23
  3.3 Result of Research Analysis ....................... 24
  3.4 RoadMap for finding the Framework ............... 25

4 Creating Evaluation Criteria ................ 27
  4.1 Requirement for Test Automation tools .......... 27
    4.1.1 Support for Various Software Testing Types 27
    4.1.2 Creating and Driving Test Case ............... 28
    4.1.3 Support of Mobile App ....................... 28
    4.1.4 Support for Operating Systems ............... 29
List of Figures

2.1 Verification and Validation ............................................. 7
2.2 Software Development lifecycle ....................................... 9
2.3 V model ................................................................. 13

3.1 The Project V model .................................................... 22
3.2 Graph for Effort and test ............................................... 25

4.1 A requirement Specification -I ........................................ 32
4.2 A requirement Specification -II ....................................... 33

5.1 A list of framework ...................................................... 36
5.2 Comparison Matrix parameters for framework list ................... 37
5.3 Assignment of Weight .................................................. 45
5.4 A framework with rating ............................................... 49

6.1 Adding script by Recording ............................................ 56
6.2 Adding script by Keyword Test ........................................ 58
6.3 Adding script by keyword actions ..................................... 58
6.4 A complete keyword test .............................................. 59
6.5 The Object Browser ..................................................... 59
6.6 The Code view for test .................................................. 60
6.7 Running a Keyword test ................................................ 61
6.8 Running a Script from Code .......................................... 62
6.9 The report format ....................................................... 63
6.10 The report format in Browser ......................................... 65

A.1 The Comparison Matrix-I ............................................. 80
A.2 The Comparison Matrix-II ............................................ 81
A.3 The Comparison Matrix-III ........................................... 82

A.1 The KT Decision Analysis-I .......................................... 90
A.2 The KT Decision Analysis -II ....................................... 91
List of Tables

3.1 Manual Test Effort .................................................. 23
3.2 Automation Test Effort ............................................. 24
1 Introduction

Mobile devices are emerging and have become more complicated with a range of features and functionalities. Many applications that were initially deployed as desktop applications or web applications are now being deployed to mobile devices. Mobile applications are getting more and more crucial in our time. The quality of these applications is one of the main objectives for companies, which have developed these applications and provide them to the user. Testers in the quality assurance departments of these companies are liable for assuring the benchmark level of software quality. In this thesis, a mobile application is described as an application running on mobile devices and taking in input information. These are either pre-installed on phones during manufacture or downloaded from an application store or through other mobile software distribution platforms.

1.1 Motivation

The company Robert Bosch GmbH deals with a range of electric and laser device tools. The company have a subdivision: PT-MT which stands for Power tools Measuring tools. The example include drills, GLM and PLR distance measuring tools, Thermo detectors etc. The subdivision of PT-MT is PT-MT/ ELF which is the engineering team that is behind the development of these devices. The department PT-MT/QMM is the quality department responsible for testing and maintaining the quality of such devices. The department introduced Do It Yourself(DIY)-PLR devices which can measure distance using laser techniques. These hardware devices can be connected to the smart phones via Bluetooth technology.

With the advent of technology, there was a need to introduce mobile apps which can connect to these measuring device. Thus the engineering department developed a new application PLR measure&go App for Android and iOS in year 2015. The apps works as a distance measurement for importing the measurement from the hardware device PLR30C, PLR40C and PLR50C. The measurement can be used to calculate area, length and breadth of figure etc. These apps are available in Apple App store and Google Play store respectively.
Chapter 1. Introduction

The department PT-MT/QMM unlike traditional testing, test these mobile apps manually. Testing such mobile apps require special test cases and techniques. The large variety of mobile technologies, platforms, networks and device poses a challenge when developing adequate strategies to test mobile software. The knowledge and experience about testing these mobile applications in general changed for different test. This mixture is good for accomplishing a high coverage of test case and finding as many failed tests as possible.

It is clear that although there is an extensive system testing carried out, it is done by a manual technique. This needs time and is less efficient. An efficient test structure for system testing is not there yet. Rather, it solely depends on the system tester to check any difference in behavior and if there is a miss in detecting the error the bug can go undetected. The time and expense, which arises in this manual testing, gives cause for a more effective and more versatile solution like Test Automation to be researched.

1.2 Problem Statement

The department PT-MT continuously develop new devices and applications concerning that device. With the introduction of the new devices PLR 30C these instruments were gradually fitted with Bluetooth interface. Thus the measurement from these devices can be transferred to mobile devices and further processed there. For the further processing of measurement, mobile applications are designed and developed within PT-MT. Examples include the Apps "PLR measure&go", "GLM measure&document" or "GLM floor plan". They are available for Android in Google Play store and iOS in the Apple App Store. These apps have been tested in their development life cycle in a manual way. That means before its launch there was an elaborative testing and GUI testing performed by the quality engineers. These tests were based on the Test plan in form of Excel table which are derived from the Acceptance criteria, a document which is approved by the Quality Management department.

During the development phase of the app, this app is tested by developers using Unit and Integration testing. If these tests are passed it undergoes a Integration Test also internally in the engineering phase. If a failure occurs the app is reworked by developers from the engineering team. If these test are passed then they are forwarded to Quality management for performing System testing which is till now a Manual approach. The manual testing approach is time consuming and is less efficient. An efficient test structure during system test is not there yet. Rather, if the error arises during the System test it is reported and the development team has to fix before the further testing can go ahead.

The time and expense which arises in this manual testing make the project delay and is a problematic for both developer and testers. This gives us a cause for a more effective and more elegant solution to be researched and find a better approach to test these applications.
1.3 Aim of the Master Thesis

As discussed in earlier chapter about the problem arising in the present project lifecycle for testing mobile apps. There must be something to be done here to find a better solution which can address this problem and help testers get their work done effectively under the limited timing constraints. Therefore, the department wanted to find a solution with which the manual testing can be minimized and most of the testing can be done by automation testing.

We have to bear in mind that we have to keep the project life cycle fixed and design a systematic test structure to be performed which later on continue for other mobile application ensuring the same quality throughout.

Test automation can be achieved by researching existing frameworks in market that can fit into the requirement of the project. However, there are many solutions and approaches available to implement. The aim of this work is to evaluate existing frameworks and finally select one of them. This is intended to find the best solution for the department and the department can continue later to use the selected Framework in the Test procedure which is performed after development of a new mobile applications.

1.4 Structure of Thesis

The remaining chapters of this Thesis are as follows. Chapter 2 discusses the Basics of Software testing and Mobile Applications. The chapter has focus on work that has previously been done either in this area or work that parallels the thinking in this thesis. Chapter 3 presents the details of scenario of a project where Test Automation has to be researched. Chapter 4 focuses on the Requirements for a framework to fulfill for fitting in a project. A pilot study that was conducted using a set of interviews with Development and Testing team to find their needs. The Evaluation criteria to select a framework is presented in Chapter 5 together with the KT Decision analysis that helps to select a framework which best fits the Requirements. Chapter 6 focuses on the Implementation of the selected framework and the documentation created for using the framework. A final summary, concluding remarks, and suggestions for future application in requirements engineering close this thesis in Chapter 7. Following the last chapter are appendixes and references in Bibliography.
This chapter deals with the themes for understanding of concepts for the entire work which are necessary. First, the term software testing is defined and illustrated by a difference of Verification and Validation strategy of test. Thereafter we understand about the Mobile Applications and the types of mobile apps. The third section explains how the testing for mobile apps is done. These form the basis for the subsequent subsection test automation. It carried out the comparison of testing using Manual and automatic approach, and benefits of Mobile Apps automation.

2.1 Software Testing

Software testing is an internal part of software development process. Let’s see what cannot be a definition for software testing. So testing does not include

- Development: If testers are able to write code, which may include tests development as automation of test include writing scripts. Inspite the fact that testers are writing code still testing — is not a development process.

- Analysis of requirements specification: During the testing process the requirement are either specified more exactly or they have to be analysed. This can be done as a necessity but its definitely not the part of testing.

- Management: Even though many companies have management position for test process, certainly controlling of test engineers is required. This also cannot be considered testing.

- Technical writing: Test engineers have to document all the test activity. This is also not testing.

To reflect the changes in software testing, we can show how the definition of testing has evolved in time:

(a) 1980: The process of executing an application in order to find errors [Myers, 1980]
(b) 1990: The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [IEEE standard 610.12-1990, 1990].

(c) 1990: Testing is not a process. It is intellectual discipline aimed at obtaining reliable software without unnecessary efforts on its control [Beizer, 1990].

(d) 1999: Testing is defined as a technical investigation of the software in order to obtain information about its quality from stakeholders’ point of view [Kaner, 1999].

(e) 2004: The IEEE model to Software Engineering Body of Knowledge defines testing as checking compliance between the real program behavior and its expected behavior on the finite set of random tests [SWEBOK, 2004]. From this point of view, anyone can say that from the year 1980 to 2004 the base theory differed so deeply that the aspect of the topic has changed fully. Definitions provided by Myers, Beizer or Kaner have described software testing as an activity aimed at something. While in SWEBOK it is defined what is testing activity, but there is no information about the aim of testing.

(f) 2015: Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test [Wikipedia- Software Testing]

In this thesis the term software testing definition is derived from [Taipale, 2007], as it is not related with any particular life cycle model or a development method:

"Testing is verification and validation."

Both verification and validation are actions which have a common goal of software quality control and error detection. Even though they have same goals, they act different in sources of properties, rules, and restrictions that are being checked during these activities.

Verification is concerned with the conformance between artifacts that are developed during the development of software. Artifacts that are created before or which are taken as input data such that these artifact comply with the standard and rules. The figure 2.1 shows the process of Verification and Validation in a development cycle.

A verification definition is provided by the IEEE standard [610.12-1990 1990]: Verification is the process of evaluating a system or a component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Validation checks conformance of any artifacts, which are being created or used during development or maintenance, with user or customer needs and requirements.

The validation’s definition is given in IEEE standard [610.12-1990]: Validation is the process of evaluating a system or a component during or at the end of the development process to determine whether it satisfies specified requirements. Verification answers the question: are we building the product right? Validation answers the question: are we building the right product? [Boehm, 1979].
2.2 Mobile Application in general

A mobile application is usually a set of program that is created to run in mobile devices such as smartphones and tablet computers, and are commonly called app. Usually apps are smaller and sole program with small functionality like a calculator, a game or web browser. The objective of these apps is to give users of mobile devices the same benefits as on normal PCs. [Janssen 2007]
2.3 Types of Mobile Apps

Mobile Applications can be divided in three categories:

• Native Mobile Application

• Hybrid Application

• Web Mobile Application

Native Mobile App:

Native App are created for use on a distinct platform or device. A native mobile app is written in a specific programming language, such as Objective C for iOS operating system and Java for Android operating systems. Native mobile apps gives fast performance and are very reliable. They can also have access to a phone’s various connection, like camera or contact list. Native app can also include gestures and can work offline. They also have access to device notification system. Usually people download native mobile apps from app stores such as the apple app store or the Play store in Android. A Native app is particular for single mobile operating system: iOS, Android or Windows.

Web Mobile App:  Web App are stored on a remote server and delivered over the internet through a browser. Web apps looks like they are unreal but they are just webpages that, in a way, look alike native applications. They are written in HTML5 and run over a web browser. Users can have access to them as they will access any web page. The process to access Web apps is simple- Just navigate to a URL and it will have option of “installing” them on their devices. The user can also create a bookmark to this page. Web apps became famous when HTML5 came around and people thought that they can have native apps features in the browser. If we look at today many sites use HTML5, thus the difference between web apps and regular web pages is difficult. Mobile web apps can be designed to run really well by most of smart mobile web browser. This includes Safari for Apple devices and Chrome for Android browsers.

Hybrid Application A hybrid application mixes aspects of web apps and native app development. The developers have been combining the features like technologies and platform independence of Web apps with the capability of native apps. The benefit of such approach is we can have the combination of both technology and there would be no fault of each technique. The main focus of this thesis is on hybrid applications as application under test is a Hybrid App. These app are really complex and it have been tough phase for the testers in these companies, which makes such hybrid applications.
2.4 Software Development Process

This section discusses about Software Lifecycle. Later on there will be concrete discussion on Basics of Manual and Automation test and why Test Automation is useful over the Manual Test approach. Lastly the chapter discuss about the benefits a project can gain if software test Automation is implemented.

2.4.1 Software Lifecycle

Software Development Life Cycle is often abbreviated as SDLC. It is also referred as Software development process. SDLC is a process detailing all the tasks performed at each instance in the software lifecycle process. There is also an international standard [ISO/IEC 12207] for software development lifecycle. This sets the standard for all the work required for developing and maintaining software.

What is SDLC?

All software organization follow the process of SDLC for a particular software project. The process includes details of the plan, how can a software be developed, maintained, replaced or be modified for new requirement. By following the defined lifecycle process, quality of software is enhanced and it adds to the complete development process.

The following figure shows the graphical representation of SDLC and its various stage

![SDLC Diagram](image)

Figure 2.2 – The stages of Software Development lifecycle
Chapter 2. Fundamentals

A typical Software Development life cycle consists of the following stages:

Stage 1: Planning and Requirement Analysis: Requirement analysis is the most important and fundamental stage of SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas. Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements: The next stage in SDLC is defining the requirements of the product and getting the approval status from client or market analysts. This is achieved by preparing a Software Requirement Specification document where all the product requirements are designed and developed for a project.

Stage 3: Designing the product architecture: Software Requirement Specification document acts like a guide for product architects for making the right architecture of the product to be developed. Depending on the Requirement document, there may be more than one design that are put forward and written in Design Document specification. This document is reviewed by the stakeholders depending on different criteria like risk assessment, product robustness, design modularity, budget and time constraints. All these criteria are responsible for accessing the best design. [Mitchell H. Levine]

A design document describes about all the modules of architecture of application along with its communication and data flow representation with the external modules. The very detailed information is specified in the DDS document which consist of internal design and proposed architecture of all modules.

Stage 4: Building or Developing the Product: As the name suggest the actual development of product starts here. The programming of the software is done and a code is generated in parallel with the Design document. Code generation can be easier if the design is well constructed and detailed, thus making it less problematic for the further steps.

Developers can follow the coding standards as defined by their companies and can take advantage of programming tools which consists of compilers, interpreters, debuggers etc which are useful for generation of the code. Different programming languages like C, C++, Pascal, Java, and PHP are used for coding. The programming language is best selected depending on the type of software being developed.

Stage 5: Testing the Product:

This stage is subdivision of the stages from the modern SDLC models because the testing procedure is involved in all the steps of SDLC. Eventually , this stage is used for testing only the
product where defects are found, tracked, fixed and retested, until the product have achieved
the quality as defined in the SRS document.

Stage 6: Deployment in the Market and Maintenance : As soon as the product is tested
successfully it is ready to be deployed and released in the market. For some of the software
releases, the deploying takes place in stages. This all depends on the business units.

The product is first released for a small segment and tested in real user environment called
User acceptance testing. Depending on the success of the deployment and feedback of small
segment, the complete product is launched with the enhancement for that particular segment.
After the official release of the product in the market, the maintenance starts for that particular
product and the customers related to that product.

SDLC Models

There are so many software development models defined and designed which are followed
during developing a software. They are often called Software Development Process Models.
Every process model pursue a sequence of steps identical to its type, so that the process of
development can be ensured successfully. [Kan, S. H. (2004)] Below are the few development
models followed in developing a software:

1. Waterfall Model
2. Iterative Model
3. Spiral Model
4. V-Model
5. Agile Model

There are some more models like Big Bang Model, Rapid Application Development and
Prototyping Models also.

Waterfall Model

The Waterfall Model was initial Process Model to be presented. It is often called as a linear-
sequential life cycle model. It is quiet simple to get a understanding and use it in project. In a
waterfall model, every stage of model must be finished before the next phase can start and
there is no overlapping in the stages.[Dr. Winston W. Royce ]

Waterfall model is the oldest SDLC approach that was utilized for software development.

Iterative Model

In Iterative model, the process starts with a simple application of a limited set of the software
requirements and slowly enhances the next versions until the final system is implemented and ready to be released. [Wiki: Incremental Model]

An iterative life cycle model don’t try to start with a complete specification of requirements. In a way, the development begins by specifying and implementing just small part of the software, which is then analyzed in order to identify further requirements. The process is rerun, making a new version of the application at the end of each iteration of the model.

**Spiral Model**

The spiral model adds the concept of iterative development with the systematic, controlled points of the waterfall model.

It allows for release of the product in a incremental way, or incremental filtration through each iteration around the spiral. [Wiki: Spiral Model]

**Agile Model**

Agile model is an addition of iterative and incremental process types with its core point on process adaptability. This is done by quick delivery of finished product.

Agile Methodology cracks the software into small increment builds. These builds are presented in iterations. Each cycle duration is from about one to three weeks. [Wiki: Agile Software Development] Every iteration involves cross functional team working together on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

Towards the completion of iteration a finished product is shown to the customer and important stakeholders.

**V Model**

V-Model is an expansion of the waterfall process and is based on connection of a testing phase for every corresponding development stage. This resembles that for every individual phase in the development cycle there is a directly connected testing phase. This is a very controlled model and user can go on next step only after completing the present step.

**V- Model design**

In a V-Model, the testing phase and the development phase are planned in parallel corresponding to individual stage. So this means there are Verification phase on one side of the ‘V’ and Validation phase on the other side. Coding phase forms the point of joining the two sides of the V-Model.

Following are the Verification phases in V-Model:

- Business Requirement Analysis: This is the first step in the development cycle of the V model where the product requirements are understood from the customer point of
2.4. Software Development Process

Figure 2.3 – The Verification and Validation stages in a V model [Robert half]

view. The process involves detailed communication with the customer to understand his requirement and what is expected from the product. It is a critical activity and a great care has to be taken for managing this activity as mostly customers dont know what are their needs.[Galin, D. (2004)] This stage also is useful for making design planning for acceptance testing as the input for UAT is taken from business requirement.

• System Design: As soon as we have the product requirements which are clear and written in detail, we can start designing the complete system. Designing a system will constitute steps like having a clear understanding and details of complete hardware and communication set up for the developed product. Also this is the stage where the test plan for System testing is created, thus saving a lot of time for executing the test at later stages

• Architectural Design: Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD). The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.
• Module Design: The detailed internal design of all the system modules is specified in this phase. This is also called Low Level Design (LLD). It is worth to be made sure that the design is appropriate with respect to the other modules in the system architecture and other external systems.[Gao, J.Z, Tsao J. H.-S, Wu, Y. (2003)]. Unit tests are a necessary part of any development process and helps removing the maximum faults and errors at an early point. Using this low level designs the Unit test can be easily written in this stage.

• Coding Phase: The real coding of the system modules designed in the design phase is taken up in the Coding phase. The programming language that fits the need is determined based on the system and architectural requirements. The coding is carried out keeping in mind the coding standards and guideline. This code goes through various reviews and is advanced for best performance before the final build is stored into the repository.[Demillo, R.A, McCracken, M. W, Martin R.J, Passafiume, J.F (1987)]

• Validation Phases: Following are the Validation phases in V-Model:
  
  – Unit Testing: During this phase of validation the code is executed according to the Unit test case designed during module design phase. The testing at code level is helpful in eliminating bugs at a very early stage and therefore its called as Unit Testing. Although all defects cannot be uncovered by unit testing.
  
  – Integration Testing: Integration testing is connected with the architectural design phase. Integration tests are done to test the communication of the internal modules and their co existence within the system.
  
  – System Testing: System testing is linked directly with the System design phase. System tests check the full system functionality of the system under development with external systems. This also checks the communication of the system with other systems Many software and hardware compatibility problem arises here during system test execution.
  
  – Acceptance Testing: Acceptance testing is connected with the business requirement analysis phase. It include testing the final product in user environment. Acceptance tests unfold the compatibility issues with any other systems available in the user environment. It also finds out the non functional issues like performance and load defects in the actual user environment.
2.4.2 Manual and Automation Test

Basics

This research is only concerned with the system test of the application. So the approach would be to concentrate on the process of Testing done during System and Acceptance Tests. The chapter explains here the present process of Manual Testing. Further the discussion is about the Automation Test process and its advantage over the Manual Test Approach. This section will become the base for the next chapters where the Automation of test as an essential approach to the present scenario of Testing in a project is discussed.

Manual testing

This section continues about testing theory started in the previous chapter, but more from the perspective of a manual tester. The next chapter describes automated testing.

For manual testing the tester is in the role of a real user. The manual tester tries to do what a general user would do, good or bad. The tester have to follow the test plan that contains the information about testing and coverage of testing. This plan should be written before the testing starts. [Black, R. (2002)]

Exploratory testing is mostly done by manual testing. Sometimes it is all the testing which is required. In this case the tester don't have to follow some exact steps from test plan. Many a times there can be more planned test cases to be followed but normally exploratory testing unveils the interface (user interface testing) and passes through all the possible features of the product taking the help of the earlier tests and their results. In order for Exploratory testing to be successful, testers need to know their own areas very well.

In addition to exploratory testing, one more used testing form is usability testing. It uses user interaction to explore the device/software functions by testing it with general user point of view. This is one really important testing method because mostly all the test cases follow certain predefined paths. According to the Wikipedia article about usability testing, “Usability testing measures the usability, or ease of use, of a specific object or set of objects, whereas general human-computer interaction studies attempt to formulate universal principles.”

Usability testing is another form of black box testing technique. The idea is to find out how people are using the product/software in order to find the errors. [Jorgensen. P.C. (1995)]. It measures response in four different areas: efficiency, accuracy, recall and emotional response. Usability requirements must be defined beforehand. If the goals are not met after the testing then the software/device is not considered usable. The following items are measured:

- Efficiency - How long does it take to complete certain tasks?

- Accuracy - How many mistakes were done while completing the tasks?
Chapter 2. Fundamentals

- Recall - How much is remembered immediately afterwards or much later?

- Emotional response - What was the user’s impression? Would the user recommend this system?

Test automation

Automated testing or test automation can ease up testing for sometime. Test automation can be defined as using a special software to control test execution. It can be much less laborious work than manual testing and much easier to find certain defects. [Ilchenko, A., 2011] It can reduce the load of a tester significantly. The tester can leave automated testing system running overnight and check the results next morning instead of spending the whole night waiting for long tests to be finished. A computer can follow predefined test scripts very precisely. Test scripts must be created manually and they must be correctly done. Otherwise the automation program is not able to execute them. Creating the tests is the bigger work to be done. Executing is quick and can be done repeatedly. However, interpreting test results, especially very large files, can be extremely time consuming and must usually be done manually. In normal cases, test automation is used together with manual testing.

There are two approaches to test automation: Code-driven testing and Graphical user interface testing. In Code-driven testing, modules and libraries are tested with many inputs to make sure that the outputs are correct. Graphical user Interface testing framework generates different user events, like mouse gestures and clicks and validates that the behavior is normal. Graphical user Interface tester is testing the interface to ensure it is meeting the specifications. Test Automation tools provide good features for user action, gesture recording and playing, and in addition to comparing them to the expected results. [Fewster, M, Graham, D. (1999)]. This approach does not require too much development. Instead it can be used with all the applications with graphical user interface. However, these features might not be so reliable after all. If something is moved or renamed, the gesture may require test re-recording and that might add irrelevant actions to the tests.

Manual testing and automated testing cover two very big fields. In each category, definite testing methods are known like integration test, system test as well as black box testing, white box testing. [Lewis, W. E. (2005)]. Some of the methodology is suitable more to manual testing, and some are best suited for automation. Here’s a small comparison of each type, with some pros and cons:

**Manual Testing**

Manual Testing is done by human who can make mistake and thus it cannot be always reliable. As manual testing takes up human resource, it is always time consuming. If you hire human resources, there is always a time and cost invested in it. Manual technique is best suited when the testing take place once or twice and frequent testing is not required.
As manual test is done by a user, it can be more useful for tests like user friendliness or improved customer experience.

**Automation Test:**

Automation test are performed by a software tool or a script thus they are more reliable. The automation approach is faster than manual approach because it uses software tools. To implement automation an investment is required to buy the test tool. When the test are run over a regular interval of time then Automation testing is the best approach.

Automation test lacks user friendliness and real time experience as it is far from human observation.

Manual testing is best suited to the following areas/scenarios:

- **Exploratory Testing:** This testing takes into account the testers experience with the product, knowledge, logical and analytical skills, and intuition. In exploratory testing, the test specification are written in a poor way and there is a short duration specified for executing the tests. Its all about the human skills that can execute the testing process here.[Myers, G. J. (2004)].

- **Usability Testing:** As the name suggest the testing to check the product is friendly, efficient enough and convenient to use by the user. In this case, human observation is the crucial point and so a manual test is more preferred.

- **Ad-hoc Testing:** In this kind of testing, there is no single way to proceed. The testing here is not planned and it solely depends on the understanding of tester to execute test.
Automated testing is the preferred option in the following scenarios:

- **Regression Testing**: Automated testing is better suitable here as there are often code changes and the test plan has to be executed regularly in a timely manner.

- **Load Testing**: Automated testing is preferred way to do the testing efficiently when load testing has to be done.

- **Repeated Execution**: Some task where a regular execution of test is required, Automation test are more useful here.

- **Performance Testing**: When many concurrent users simulation is required then automation testing is a good approach.

Taking in mind these factors, we can find the best fitting technique for any given testing situation and achieve best result within budget and timeline.

### 2.4.3 Benefits of Automation Test

The Automation Test have following Benefits over Manual Test and this paves the path of Selection of a framework for the project where presently Manual test approach is followed.

**Save Time** – It is one of the important benefit of automation when regression tests are concerned. Regression test are done for retesting the newly introduced features in an application. It is also useful when an update is made in the present application. This change can be due to result of change request, defect fix or re-factoring. The goal of regression test is to ensure the application is working as expected after any change was made. Thus in order to check this, we need to run all test cases associated with the change. There is a danger here, that due to less time all tests associated to the change cannot be run and this may lead to undiscovered defects. [Hayes, L. G., 1996,]

This issue can be solved by setting up automation tests to run over night or after every deployment. This will give time for the tester to do exploratory testing and concentrate on things which cannot be automated and concentrate on other tasks.

**Speed** – Usually Automation test are done with the help of a tool so obviously they are faster then manual test. This add to the benefit of time savings.

**Repeatability** – When the test are run again and again they remove the chances of human mistakes like forgetting the desired action, skipping steps from test plan or knowingly not performing the right test. These all can lead to an error and thus such error sometimes goes unnoticed. Also sometime wrong errors are reported and takes time to get resolve as they cannot be reproduced by developers.
2.4. Software Development Process

**Maintenance of the test suite** – As the name goes, maintaining the test script is very essential. Sometimes it may happen that test suite is older and new functionalities are added which requires changes in the test script also. This is because the tester did not had time for updating the test scripts. If these older scripts are executed they may fail and eventually result in an error. This should be fixed by testers and he has to ensure the correct test script. Thus ensuring updated script is very important

**Reusable** – The scripts from automation can be reused later in different version of application or in different interfaces.

**Increase Coverage** – With automation, more features can be covered and helping to build the test coverage.

**Cost Reduction** – Most of the tasks are handled by software tool, so it does not require many resources to hire. Thus in a way we are reducing the cost bared during regression test.[Hoffman, D., 2007].

In a small summary, automation testing helps to reduce time taken for test execution, enhances the quality of software appliaction by being reliable, repeatable and being comprehensive to the test suite. It also utilises human resources by using their skills and time where it is required more thus incresing the test coverage.

Lately, automation tests does not rule out the role of tester. There will always be a position for a manual tester within a project team, and not every test connected with a feature can be automated Also, not every software project is good for automated testing. As a quality assurer, automation is about making tasks easier, by using automation to conquer problems like time and doing test more effectively to ensure quality is always present not just in the application being developed but also in the test process.


3 State of Art

3.1 Process of Mobile testing in a Project

The department PT-MT is of prime importance for Bosch Powertools located in Leinfelden. The engineering department make some products and tools which, by means of laser can carry out measurements. The department Quality management is responsible for the testing of these tools and the application related to that particular device.

The process of mobile application development in the project follows the combination of V model and the SCRUM cycle.

Each phase from the Verification phase forms a Scrum cycle with its corresponding Verification phase. From the V model described in section 2.4.1 the Detailed design cycle and the Unit test along with the coding phase forms a small V model in the project. This Scrum Cycle gets repeated after every tasks are completed.

Going up in the V model, the High level design and Integration test forms the higher layer of Scrum cycle in the project model. This cycle is repeated for every Sprint. [Coghlan, D, Brannick T. (2005)]

The Functional Specification and System testing forms the Highest level of Scrum cycle and is done before every release of project. The Requirement specification document and Acceptance test are part of the highest level of Scrum cycle from the System test in the project.

Each phase of the development process will have a specific input and a specific output. Once the project is confirmed to start, the phases of the development of project can be divided into the following phases:

- Software requirements phase.
- Software Design
Whenever a new project comes in the team starts working in form of a development model. Initially the Requirement definition is specified from the client. This leads to development of the Requirement specification document (SRS). This is usually acceptance criteria in our case.

From the acceptance criteria project plan is developed. The responsibility of testers is to create software test plan and project plan from this acceptance criteria. Developers start coding from the design. The project work is divided into different modules and these project modules are distributed among the developers. In meantime testers responsibility is to create test scenario and write test cases according to assigned modules. These are in general in accordance with the Acceptance criteria. Tester try to cover almost all the functional test cases from Acceptance Criteria. The data is maintained manually in an excel test case templates.

When developers finish individual modules, they test their respective modules in form of Unit or integration test. If they fail these test, modules are reworked by respective developers for fix. For passed modules manual testing is carried out from the written test cases. If any bug is found it gets assigned to module developer and get logged in bug tracking tool. On bug fix tester do bug verification and testing of all related modules. If bug passes the verification it is marked as verified and marked as closed. Otherwise above mentioned bug cycle gets repeated.
3.2 Effort Estimation for Manual and Automation Test

Before searching the right Automation framework, there is a need to evaluate how much effort it take to perform Manual testing in one application and what benefits can be taken from Automation test approach.

There was a analysis done on the previously developed mobile application "PLR measure&go" from Robert Bosch GmbH. The data was collected by interviewing the Quality team located at offshore in India. These teams are responsible for creation of Test plan, Writing the test cases from the Acceptance criteria document and later executing them during System testing.

The questions for interviewing the test team were regarding

- The Amount of time taken for writing the test plan.
- The Amount of time taken for writing the test cases from Acceptance criteria.
- The time taken for executing the test cases.
- The number of devices used to execute the test plan.
- The number of software releases for the application.

So on an average we were able to get a table concerning the amount of time taken for executing the test. It took overall 108 men hour to execute test plan for PLR measure&go with total 6 releases.

<table>
<thead>
<tr>
<th>Project</th>
<th>Number of mobile devices</th>
<th>Release 1.0.0</th>
<th>Release 1.0.1</th>
<th>Release 1.0.2</th>
<th>Release 1.0.4</th>
<th>Release 1.0.5</th>
<th>Release 1.1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLR measure&amp;go</td>
<td>6</td>
<td>34</td>
<td>48</td>
<td>62</td>
<td>76</td>
<td>90</td>
<td>108</td>
</tr>
</tbody>
</table>
Chapter 3. State of Art

After calculating the total effort spent for taking the Manual test with PLR measure&go application we have to see how the Automation Test approach would continue. For this we assumed that the initial effort for Creating test case and test script for Automation takes nearly twice the time it took for Creating the test case and test script. We have to keep in mind that the test set up of tool and working on scripts takes a longer duration as compared to manual test where these steps are not involved.

Table 3.2 – Automation Test Effort

<table>
<thead>
<tr>
<th>Project</th>
<th>Number of mobile devices</th>
<th>Release 1.0.0</th>
<th>Release 1.0.1</th>
<th>Release 1.0.2</th>
<th>Release 1.0.4</th>
<th>Release 1.0.5</th>
<th>Release 1.1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLR measure&amp;go</td>
<td></td>
<td>6</td>
<td>54</td>
<td>59</td>
<td>64</td>
<td>69</td>
<td>74</td>
</tr>
</tbody>
</table>

From the automation testing we assumed after the initial high effort, the next releases would only involve using the already available script to create new functionalities. [Bain, J. Lee (1978)]. Thus the next effort are taken less as compared to Manual Test. Obviously, this can be a different number when Automation test is actually implemented.

3.3 Result of Research Analysis

The result of the research analysis can be shown in a form of a graph of Effort vs Test releases from the tables given in 3.1 and 3.2. The Blue line shows the Manual test effort to develop the application from Initial to Final release. The Red line shows the Automation test Effort from Initial release 1 to Final release 5.

The Manual test effort rises Steeply while the Automated Test Case effort increase moderately. This can help to find the Break Even point. This point is a place until which Manual effort increases steeply while the Automation Test effort goes down.

The test managers can conclude from this point that Manual and Automation Test Efforts are more or less similar till this point. It is from Break even point the Automation Test Effort gives return on investment and the team saves Effort.

From the graph it gets cleared that roughly 29 men hour can be saved using Automation Test Approach. So any test effort saved would be seen after the Test Release 3 has been completed. Before the Test Release 3, the Automation Test will take more effort. The graph for the Manual and Automation approach is shown below:
3.4 RoadMap for finding the Framework

To achieve the objective described in section 1.3, there will be discussion to understand what are the requirements expected from the Automation tool that should be fulfilled in accordance with the current scenario. This is discussed in Chapter 4. The discussion would be very brief on what kind of testing should be supported and what devices support should be available. Next a evaluation will be done by creating a comparison matrix. Based on the evaluation there will be filtering of frameworks and the explanation of why this framework cannot be considered further. On the second phase of evaluation, the framework available will go through a low level research by taking a trial of the tool and implementing the application with the test environment.

According to the behavior shown in trial a Kepner Tregoe commonly called KT Decision Analysis will be done where each framework after taking trial will be given a score and the framework with the highest score will be Selected as the framework for implementation in the complete test process of System testing. Lastly a small documentation for further usage of this framework and all the set up procedure will be specified in a document.
Determination of requirements:

To find the optimal solution to the problem described in Chapter 1.2, a detailed evaluation criteria has to set up. This can be achieved by creating a requirements specification document in the first place. The evaluation criteria is determined by using the survey and creativity technique. There were meetings held with the department staff and quality engineers to discuss the actual requirement.

The following are the requirements to be defined. The full Requirements specification can be found in Appendix A.

4.1 Requirement for Test Automation tools

4.1.1 Support for Various Software Testing Types

The Framework has to support:

- System testing

System testing is done to make sure that by keeping the system in another environment like with a different operating system, it still performs as expected. System test are done under a complete test environment and all the function of an application are implemented. This type of testing is a part of Black Box testing.

- Regression testing:

  Regression testing is done when a software is changed or any update to present software is done. This kind of testing is performed to evaluate if any other part of software gets affected due to changes made. Also whenever a bug fix occurs, regression tests are done to make sure that any other function or requirements from business does not get affected.

- Acceptance Testing:
Chapter 4. Creating Evaluation Criteria

This is the crucial part of testing as it is done in the user specific environment. It is performed by the Quality engineers from the user end.[Parrington, N. (1989).]. Acceptance test are done to verify wheather the software meets the requirements from specification documents. The test team will have some Acceptance criteria that will be useful to test the application.

4.1.2 Creating and Driving Test Case

The Framework has to support:

o Linear Scripting

This is the simplest of all frameworks, and very similar to the Record-Playback mechanism. In linear scripting every details about the user interaction with application is saved in a script. It is actually a script created when a manual tester performs testing and interaction are supposedly recorded.

o Code- driven testing

Code driven test automation is a useful aspect of agile software development, where its called as Test-driven development (TDD). In this aspect of testing before the code is written for testing a unit test is written to define the functionality of code. When all test have passed then this code is considered complete. In our case that can be after the source code is written.

o Data-Driven Testing

A Data-Driven test framework indicates the repeating of the same test case scenario using different data sets. This make the test case more data centric. A Data-Driven script is parameterized, changing hardcoded values with variables, in which the data may be loaded from a number of sources, including: text files, Excel files, CSV files and databases.

o Keyword driven testing

It is an approach in which the control to verify and execute Test cases is maintained with external data files. Thus the test data and the operations and sequence of the test is planned in external data file. An extra library is required to interpret this data eventhough a conventional script is already present. It is an extension of data driven testing.

4.1.3 Support of Mobile App

The framework should support

o Native apps: These apps live on the device and are accessed through icons on the device home screen. Native apps are installed through an application store.

o Hybrid App: Hybrid apps are part native apps, part web apps. Like native apps, they live in
4.1. Requirement for Test Automation tools

an app store and can take advantage of the many device features available.

4.1.4 Support for Operating Systems

It's not necessary to find a framework which supports both OS Android and iOS. But it's good to have Single framework which supports both OS.

4.1.5 Automated Test Scripting

The Framework has to support:

- Manual Scripting:
  It means that the Tester can write a script in a standard language using standard languages, such as VBScript or JScript rather than their dialects. This will decrease the time needed to get acquainted with a product

- Record/Playback function:
  Record/Playback function means that the tester is able to record a test case by executing the App on a device or emulator. The script is automatically written by the framework based in the commands so that the tester doesn't have to write a single line of code.

4.1.6 Scripting Support

The Framework has to support:

- Single Script compatibility:
  It is very preferable that the scripts created for iOS and Android are compatible / similar. It is recommended that a single script is used for executing on both iOS and Android devices without much change.

- Keyword Script View:
  Preferably the Script View is keyword-driven. That means that there are abstraction layers so that also non-programmers can understand the scripts by looking at it. This means there must be a view for Programming language specific understanding like Jscript or VB Script.

- Object Identification:
  The tool should have a powerful object identification capability. An Object is any entity which can be located in the app. It can be a button, dropdown menu or a cross for closing the app.

Support for all object identification methods should be supported. The tool should be able to identify an object by its Native ID, Image Recognition, Text Recognition and Web HTML5.
Chapter 4. Creating Evaluation Criteria

This is crucial otherwise the maintenance of the tests will be very poor.

4.1.7 Test Execution

The Framework has to support:

o Test Run on Devices

The framework has to support the test execution on at least a local emulator or a local device. Otherwise an emulator in cloud or preferably it gives us the opportunity to use several devices accessible over the cloud.

o Multiple/Parallel test Run

There must be possibility of testing device in parallel from one machine. We should be able to create our test once and then play it back from one machine on all the available devices at one time.

o Defect Logging:

It’s important that the framework should report failed results and they can be logged in the present bug tracking tool. So, there should be a connectivity between the automation tool and Bug tracking tool.

o Test Reporting

Standard report Format: The data format for the standard report of the framework should be descriptive. The report format should be detailed view with possibility of screenshot of the steps executed. Also, in case of fail results the failed steps can be located in the script.

4.1.8 Configuration effort:

The configuration of the framework should be connected with an acceptable effort. This parameter will be more decisive in the end of the framework selection.

4.1.9 Pricing Policy

The framework has to support:

Costs:

The framework can be commercial licence tool or a open source. Until the requirements are fit by a framework there is budget for buying the license for even a commercial tool. If paying for a license there should be a comparison for which license fits the need.
4.2 Prioritization of Requirement

**Structure:**

If the selection is for a commercial tool then, the framework should be checked for the licensing structure available. The license should be checked for Single node license or a floating user license is required. Also, the license fees has to be paid monthly or yearly can also be a matter of research.

**4.1.10 Maintenance:**

The Framework has to support:

- **Assurance by provider:**
  
The Maintenance should be assured by the provider especially if we have to buy a commercial tool with a license. The support should be either over the email, on call or in person depending on the critical condition of the requirement.

  Also, we should have no big adjustment needed on our side to change the project specifications.

- **Recent version:**
  
  If the framework is open source the last update has to be at least in 2014. Preferably it is in the actually month.

- **Ease of support:**
  
  There have to be a support available to make sure that we can solve upcoming problems with the framework. For that there should be at least enough online learning material like tutorials etc. or a community.

With this information about the important requirements a test automation tool should satisfy, a detailed list of requirement specification would be attached in appendix A. The next chapter discusses how can prioritization of requirement can be done.

**4.2 Prioritization of Requirement**

This chapter describes the requirements defined in 4.1 which now have to be prioritized. This step is essential for the subsequent selection of the framework. As a prioritization process, there is a standard defined by IEEE called Standard 830-1998. It is based on one criteria classification. According to this, the "requirements are Important for the implementation of the framework and the success of System."

For all the requirements collected from the project point of view it was necessary that these requirements are grouped according to a priority. This will be easier to understand requirements
which cannot be ignored. Thus it will be helpful for selection of a framework and further success of the System test. [Plotytsia, S., 2014]

There are three priority classes defined in the Requirements document:

- **Mandatory Requirement**: Requirement that have to be met unconditionally.
- **Recommended Requirement**: These are an important requirements, but lacking some features does not effect the success of output.
- **Optional requirement**: Requirements whose absence does not effect success of the output. They have the minimum liability to a project.

The below figure represents the parameters and Criteria from the requirement specification document and the priority assigned to that in the next column.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Criteria</th>
<th>Priority of Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for Various Applications</td>
<td>Support Programming tools</td>
<td>Recommended</td>
</tr>
<tr>
<td>Platforms</td>
<td>o Eclipse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o Xcode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o Android Studio</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version control tool:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tortoise SVN / Subclipse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Continuous Integration tool</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o Jenkins</td>
<td></td>
</tr>
<tr>
<td>Support for Operating Systems</td>
<td>iOS OSX</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Android</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Support for Mobile App Types</td>
<td>Native apps</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Web Apps</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Hybrid App</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Supported Testing Type</td>
<td>System Test</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Regression Test</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Acceptance Test</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Integration test</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Unit Test</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>GUI Testing</td>
<td>Recommended</td>
</tr>
<tr>
<td>Creating and Driving Automated</td>
<td>Linear Scripting</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Test Case</td>
<td>Code Driven Test</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Data Driven Test</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Keyword Driven Test</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Modularity Driven Test</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Style Guide test</td>
<td>Optional</td>
</tr>
<tr>
<td>Automated test Scripting</td>
<td>Manual Scripting</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>UI Aided Scripting</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Record Playback function</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Figure 4.1 – The Requirement with the parameter definition and their priority Part-1
### 4.2. Prioritization of Requirement

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Criteria</th>
<th>Priority of Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scripting Support</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Single script compatibility</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Keyword Script View</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Support for Editor</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Object identification</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Debugger Support</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Script Execution Speed</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Interface Versioning</td>
<td>Recommended</td>
</tr>
<tr>
<td><strong>Test Execution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Run on Device</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Multiple Test Run</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Test/Code Coverage</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Defect Logging</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Test reporting</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard Report Format</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Report Customizable</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>Exporting Report</td>
<td>Recommended</td>
</tr>
<tr>
<td><strong>Configuration Effort</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td><strong>Pricing Policy</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Structure</td>
<td>Recommended</td>
</tr>
<tr>
<td><strong>Maintainence</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assured by provider</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Recent Version</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Ease of Support</td>
<td>Mandatory</td>
</tr>
<tr>
<td><strong>Free Trial</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.2 – The Requirement with the parameter definition and their priority Part-2
In this chapter there is a choice of a framework based on Chapter 4 where we defined requirement specification. There are some existing Frameworks which are first collected and evaluated. The evaluation is divided into two steps:

- Collecting list of framework

With information gained from market, peer to peer recommendations, already available frameworks or company specific automation tool, all frameworks should be collected and written down. [Dhall, S, 2008.] Any framework that has anything to do with automation test have to be included in the list of framework and later on we can compare these tools with already present Requirement Document.

- Preparing matrix for all possible combinations

After defining parameters of importance, a matrix representing the results for each combination, is prepared. The result is a product of parameters and framework. The criticality of this combination is proportional to the result. A high match indicates high criticality. Based on the criticality of the combination, the matrix can be used to determine the few frameworks which are supported.

### 5.1 Collecting list of framework

In order to find common frameworks for automation testing there was an initial research on internet about the frameworks already available in market. Also from the last implementation of Unit test framework in the project there was already some framework available to consider for System Testing. In order to get frameworks for automation testing all frameworks which have anything to do with it were collected and written down in a list. [Hughes Systique Corporation, n. a., 2013.]
Chapter 5. Evaluation of framework

The frameworks that have been added to the list are shown in figure 5.1:

![Figure 5.1 – The list of collected frameworks](image)

5.2 Creation of Comparison Matrix for all possible combination

The list of framework collected in Section 5.1 are now evaluated in a comparison matrix against the parameter requirement specification. These parameters form the Basis for comparison matrix. In the below table (figure 5.2) the user can find the Requirements criteria from the Requirement specifications discussed in chapter 4.1. These are used as parameters and each framework is compared accordingly. The parameters from requirements specification document have been written in the rows.

The upper third column of the matrix gives the framework names. Every framework that has been collected in the section 5.1 will be written in a subsequent column. Thus, each framework must be tested for each individual request. Furthermore, this constitutes an overview that can be helpful to view the frameworks quickly. Based on this comparison the list can be handy to filter the frameworks which are no longer acceptable in the selection criteria. The complete comparison matrix can be found in Appendix B. The information which are necessary for completing the matrix are derived from online materials such as the Framework site, forum posts or tutorials. All reference to sources that were used to fill in the matrix are given in Appendix C.
5.3 Filtering of Framework

The chapter deals with the operation that was performed to filter out the framework which did not meet the requirements criteria. Overall 18 frameworks were ruled out of further considering the framework as they did not meet the condition specified in the Requirement Specification document. The priority of the parameters which were 'mandatory conditions' have to be completely fulfilled for a framework to go for further selection.

![Figure 5.2 – Comparison Matrix parameters for framework list](image)
5.3.1 Reasoning which frameworks are not supported

The filtering of frameworks was done in presence of the engineering team and quality engineers and the reason for filtering the particular framework was completely agreed on. The following framework are no longer considered for further research:

1) Abbot: The last release was made in 2011 and the framework does not support System Testing which is a mandatory requirement.

2) Concordian: The framework does not support data driven testing. Also Regression test will not be possible with Concordian.

3) Espresso (Android kit): The framework does not support Data driven testing and Keyword driven Testing. The requirement is also not fulfilled for Regression test.

4) Fitnesse: The requirement is not fulfilled for the points Regression Testing. The requirement Compatible Operating Systems is not fulfilled in case of iOS. Also The test execution request is not met in the point’s emulator and connected device.

The requirement is not supported for the point Linear scripting and Keyword driven Testing from Requirement Specifications.

5) Instruments: The framework does not support System and Regression testing. There is also no provision for Data and Keyword Driven tests.

6) IOS Driver: The framework does not support System and Regression testing. Not suitable for Data driven Testing and Keyword driven testing. For using Android operating system a different framework has to be utilised. For eg: Robotium.

7) J Unit: The Framework does not support Regression testing. The support is not for Keyword Driven testing.

8) MonkeyTalk: The framework is No longer available to work on until next 6 months, after acquisition from Oracle.

9) Meux Cloud: A similar option Perfecto mobile being considered.

10) M3: The framework does not support Hybrid Apps. There is no provision for Regression testing. There is no support for Keyword Driven testing.

11) Native Driver: It supports only for functional System testing and it is not suitable for Regression tests. It is Only script based, so cannot allow UI aided script or record playback procedure.

12) Selendroid: It supports only Android operating system. Support API level from 10-19, so it is not good for Android devices working on API level greater then 20. A Similar approach for testing is using Appium which can be considered. It also have support for further API level
and even iOS is supported.

13) Silk Mobile test: It is difficult to write test in product specific language 4test. There are better options available if we consider a commercial tool.

14) SOASTA Touch test: The framework does not support Manual Scripting of test case. It is often difficult to set up device and apk into the browser. It is not suitable for System and regression test. There is also no provision for Data driven testing.

15) Test object: It’s not actually a test tool, just a device cloud service which uses Robotium, Appium and Espresso as frameworks to automate test. Instead using this, we can in house download any of the framework and can test in the device available with us.

16) Test Flight: The framework does not support system and regression test. It’s actually an online service for over-the-air installation and testing of mobile applications, not actually a framework tool. No support for Keyword and Data driven test. Also there is no support for automating script manually.

17) Keynote Device Anywhere: It is a Cloud-based solution, but it requires hard wiring of devices for use. Cloud service is slow and not super responsive. It is Expensive then a similar option which is Perfecto mobile.

Thus after a first round of filtering following frameworks were selected for further consideration.

• Appium: It meets all the requirements from Mandatory parameters. It works just like Selenium web driver. Also many scripting languages are supported by it which makes it easy for developers to work on it.

• Calabash: It also meets all the requirements from parameters which are mandatory. Furthermore, this framework has its own scripting language Gherkin, which is a natural language. Thus even a non-programmer can work with this language.

• EggPlant Mobile: This framework also meets all the mandatory requirement of the parameters specified in requirement documents. It has functionality of image based testing, where each image can be evaluated with the expected image. This also have a framework specific language Sensetalk which is also a natural language.

• HP Mobile Center: The framework fulfills all the parameters from mandatory requirement. This is one of the leading tools in the world of Automation testing. The scripting language is Visual basic Scripting, which is easier to understand for a non-programmer.

• Perfecto Mobile: The framework satisfies the mandatory requirement of all necessary parameters. This is a framework which is commercial and it can be accessed over the website. It can be helpful for enhancing new features as it utilize its use over the cloud.
Chapter 5. Evaluation of framework

- Ranorex: This also meets all the Mandatory requirements. It has also a record and play feature which can be easier for setting up test. Also it supports many scripting languages so that the developers and testers can easily work.

- Robotium / Robotium Recorder

Robotium meets all the requirements of Mandatory parameters. But this framework only works with Android. However, in lower level research user can check if it can be a good framework for only one operating system. In addition, it also offers a Robotium Recorder which has a Capture and replay Method.
If the user can work with iOS driver for Apple operating system then Robotium can be a good choice for further consideration

- See test Automation:

This also satisfies the mandatory requirements of parameters written in specification document. The framework also have provision of record and play method. It also supports many scripting languages.

- Test Complete:

This also satisfies all the mandatory requirements. Also this has a feature of writing the manual scripts and recording the scripts also. This can be useful in creating test cases with little knowledge of programming.

5.3.2 Further research on remaining framework

After filtering the frameworks from initial list there will be low level research conducted on the frameworks. The procedure will be to take a trial of the selected frameworks and check the behavior with the application under test.

The research is based on three simple steps:

- Downloading the trial version of software.

- Installing the software for use and setting the test environment with application under test.

- Writing down the behavior of the software tool with the application under test and preparing a document for it.

This process is repeated for all the 9 frameworks. The result of findings is given in the documentation prepared for each of these framework which went under trial. The result of findings is shared with the quality department.
5.3. Filtering of Framework

5.3.3 Results after low level research

Following is the results of behavior shown by frameworks which went under trial:

1. Appium :
   a. Positive point: It does not require source code to start automation. Also we ship the app what we have tested. Lastly, the framework is Open source.

   b. Negative points: It supports Android API from level 17 so for previous version user have to use Selendroid. Simultaneous test run on multiple devices is not possible with Appium so the user has to run the test one by one which adds the time taken for executing tests.

   For seeing Execution results, the user has to go to Appium server and check the result which is not a good option. Also appium does not have a good reporting functionality, and generating a report is complex task. The overall Configuration effort with appium is high as the user has to create scripts in Windows and Mac systems separately.

   c. Solutions: The Sauce labs provide option to sign in the account if user needs a better Reporting features with details of test Step and Test step screen shot but this service is paid.

   d. My Opinion: The option of selecting Appium can only be done for Unit and Integration test, but for System and Regression Test there is a need of framework which supports Test Reporting and Low Configuration Effort as a major factor.

2. Calabash :
   a. Positive point: It supports both ios and Android systems. The framework is Open Source.

   b. Negative Points: For Writing the script the language is Cucumber, which is not a standard language. The scripting language is Easy to read, but difficult to implement the use as compared to standard language which have a set syntax. Calabash does not install the app in device, so user has to make sure the app is installed on device beforehand. There are some situations arises where gesture are not written in cucumber, then the user has to write the code in Ruby for getting this gesture. This can be tricky.

   There are changes required in the source code for testing the Application under test. For eg- User have to add Ruby library and cucumber gems to source code, so we have to check the behaviour of application if adding such libraries have any impact. Calabash report have to be formatted. Without specifying a formatter, the results will be printed on screen. If user have a lot of test scenarios in test suite, it is going to be painful to scroll through your terminal to spot the results.

   c. My Opinion:

   Calabash can be used for System test only if the team is open to learn natural language like Cucumber and the developer team can support the integration with Ruby language.
Chapter 5. Evaluation of framework

From the project point of view, the developers here work with standard language in Java and objective C, so selecting a tool with entirely different language is not a preferred choice.

3. EggPlant:
   a. Positive point: Setting up the test Framework is easy. Low configuration to write test case. It has a license fees.
   b. Negative Point: The language used for scripting is Sense talk which is not a standard language. Also its last release was in 2007 and later no upgrades made. Image detection is not good, user finds difficulty to detect images during test Run. Same script cannot run for landscape view and Portrait view. Different Image libraries are necessary for iOS and Android, so different scripts should be created.

4. HP Mobile:
   Positive point: Supported by a big company HP, and HP UFT is the base of concepts for all Automation tools.
   Negative Point: It cannot support Hybrid mobile app. The licensing cost is very high around $8000-$12000 which cannot be considered for present test scenario. Could not have access to Test trial the tool, so further details and shortcomings are not known yet.

5. Perfecto Mobile:
   Positive point: User does not have to use smartphone devices as the devices are in cloud. No set up required on user side, just log in with the user id in browser. During the Manual execution the reporting functionality is good, user can also have a video playback of all the steps executed. The service is paid and a license have to be taken. Negative Point. During trial, user cannot have access to Automation tab.
   During the manual execution of the service, the test tool is Slow in processing steps. Some gesture have to be scripted using the editor as they are not supporting recording. Also, User will not be able to connect to PLR hardware devices with smartphones, as the device is in cloud.

6. Ranorex:
   Positive point: The tool is easy to set up and Recording is easy. Test Reporting is good and errors can be detected easily.
   Negative Point: The tool lacks support to connecting to System apps during Recording. The tool cannot detect application pop ups like Bluetooth Connection during the test trial. Problem with Adding objects to Object Repository directly from mobile.
   The tool does not have a emulator screen, so test runs directly on device.
7. Robotium+ Recorder:

Positive points: It is a Open source tool. It Uses present developing tools like Eclipse and Android studio from the development team.

Negative Points: The solution is for Android only. The User should have access to Source code to start Automating. Initial Setting up test is a high configuration effort task from beginning. Robotium Recorder was not able to Detect system pop ups during the recording.

8. SeeTestAutomation:

Positive point: Easy set up and Creation of test script.

Negative point: There was a difficulty in binding the recorded step into exact keyword Test. Some steps were not detected while doing the test run. The recording should be done slowly as said by the support team. The code view cannot be edited in the test tool, it has to be exported to the editor of your choice for making changes in code.

Also, writing the script and running the test simultaneously is not possible. Either user has to record test step or add the keyword test to be able to script other features. Tool crashes after switch from camera to app under test is made, maybe updates can fix this issue with Seetest.

9. Test Complete:

Positive Point: The installation is easy and it contains powerful recording tool. It Can detect the system pop ups and perform operation on them. The tool Was able to connect to system apps like camera and was able to take the screenshot. It can easily perform scripting in standard language like java or Vb script.

The tool Have a better reporting function and clearly shows Test fail Steps. It Can be connected to Bug tracking tool, so test execution can be streamlined. Overall a good tool to perform System test and go with the implementation.

Negative Point: The user has to add some libraries from TestComplete and create a new package file for iOS apps. It has to be checked if this library makes any problem with the already added library files in XCode. Until recently after trial there was no problems faced after adding this library to our App. The framework is having a license Cost for further use.

My opinion: The tool looks good to go with System and regression test. The configuration effort is low and the user can create script in less time if has experience working on it. Working with iOS and Android system has to create separate scripts but the scripts are compatible across all iOS devices for iOS operating system and across all Android devices for Android operating system.
5.4 Creation of KT Decision Analysis table

When there are many factors to consider for making a decision a KT Analysis is a convenient tool for making and documenting a decision. This process is useful in the present scenario where 9 frameworks are considered and the selection of one framework has to be done.

During decision making from a range of results it may happen often that we get stuck in selecting one result due to the many factors present to be considered. This can be simpler if we get all the considerations in the decision down on paper which will surely make a huge difference. This is what is expected to be done sometimes.

When the decisions are more complicated or the decision requires to give evidence of the process and how result is achieved, a more detailed tool is required. The simple KT Analysis is the tool which is very effective and user friendly and can be considered in such cases.

KT Analysis stands for Kepner Tregoe Analysis, for the makers who originally invented the idea.

5.4.1 Steps for KT Analysis

There are three steps to work with this tool: weighting, rating, score.

1. The Factors in the Decision - WEIGHTING
   Initially, make a list of all the criteria or parameters that are crucial to the decision. When we are finished with the list, the next step is to give a weight to each parameter out of ten. The ten is the highest weight that can be allocated to a parameter because of the priority.

   In order to achieve the best result, it is crucial that the user complete this step earlier. To get the best and most objective outcome, it is critical that you complete this step before user look at the options available from the list. Thus it is encouraged to determine the parameters and their importance first.

2. The Choices - RATING
   The next step in decision making is rating where we chooses from the list of available result. The user makes a list of the alternatives and then a rating is being given to each parameter based on the criteria we wrote in Step 1. The rating is given basically out of ten, but the main idea is to rate the alternative results based on their performance of fulfilling the criteria.

3. Calculate the Outcome - SCORE
   The last step is to find the total score for each of the alternative. For this we calculate the product of ratings received by parameters and the weight allocated to that parameter. In end just add all the scores to get the total score for each alternative.

   The alternative with the highest score is the preferred choice for selection. There can be a little
5.4. Creation of KT Decision Analysis table

research on the alternative which score closely. But as a matter of fact if the weights were allocated correctly the total score will give the obvious results and the highest score is the clear selection.

5.4.2 Assigning weights

The KT Analysis table is found useful to find the framework for automation test in our case. As we have seen the frameworks which have been filtered and tested for trial, the results of the trial were stored in form of a Decision Analysis table.

- Assigning weights to the Parameters:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported app types-Native</td>
<td>10</td>
</tr>
<tr>
<td>Supported app types-Hybrid</td>
<td>10</td>
</tr>
<tr>
<td>Supported app types-Web</td>
<td>2</td>
</tr>
<tr>
<td>Supported OS-Android</td>
<td>10</td>
</tr>
<tr>
<td>Supported OS-iOS</td>
<td>10</td>
</tr>
<tr>
<td>Supported OS-Windows Phone</td>
<td>2</td>
</tr>
<tr>
<td>Supported tests-system test (Acceptance test)</td>
<td>8</td>
</tr>
<tr>
<td>Supported tests-Regression test</td>
<td>7</td>
</tr>
<tr>
<td>Supported tests-GUI testing</td>
<td>5</td>
</tr>
<tr>
<td>Supported tests-integration test</td>
<td>3</td>
</tr>
<tr>
<td>Supported tests-unit test</td>
<td>2</td>
</tr>
<tr>
<td>Code-driven testing</td>
<td>6</td>
</tr>
<tr>
<td>Linear Scripting</td>
<td>8</td>
</tr>
<tr>
<td>Data-driven testing</td>
<td>8</td>
</tr>
<tr>
<td>Keyword-driven Testing</td>
<td>6</td>
</tr>
<tr>
<td>Modularity-driven testing</td>
<td>5</td>
</tr>
<tr>
<td>Style guide testing</td>
<td>2</td>
</tr>
<tr>
<td>Writing test-Script based</td>
<td>10</td>
</tr>
<tr>
<td>Writing test-UI based</td>
<td>5</td>
</tr>
<tr>
<td>Writing test-Record/Playback</td>
<td>8</td>
</tr>
<tr>
<td>Supported interfaces-Eclipse</td>
<td>5</td>
</tr>
<tr>
<td>Supported interfaces-Android Studio</td>
<td>5</td>
</tr>
<tr>
<td>Supported interfaces-Xcode</td>
<td>5</td>
</tr>
<tr>
<td>Supported interfaces-instruments</td>
<td>5</td>
</tr>
<tr>
<td>Supported interfaces-Jenkins</td>
<td>5</td>
</tr>
<tr>
<td>Script compatibility</td>
<td>9</td>
</tr>
<tr>
<td>Keyword View</td>
<td>8</td>
</tr>
<tr>
<td>Support for Editor</td>
<td>3</td>
</tr>
<tr>
<td>Access to Third Party app</td>
<td>10</td>
</tr>
<tr>
<td>Object Detection</td>
<td>7</td>
</tr>
<tr>
<td>Debugger Support</td>
<td>5</td>
</tr>
<tr>
<td>Script language</td>
<td>8</td>
</tr>
<tr>
<td>Script execution speed</td>
<td>3</td>
</tr>
<tr>
<td>Interface versioning</td>
<td>5</td>
</tr>
<tr>
<td>Runs on Local emulator</td>
<td>8</td>
</tr>
<tr>
<td>Runs on Local device</td>
<td>10</td>
</tr>
<tr>
<td>Runs on Emulator in cloud</td>
<td>5</td>
</tr>
<tr>
<td>Runs on Devices in cloud</td>
<td>5</td>
</tr>
<tr>
<td>Support code coverage</td>
<td>7</td>
</tr>
<tr>
<td>Execution in parallel Run</td>
<td>5</td>
</tr>
<tr>
<td>Error Handling</td>
<td>3</td>
</tr>
<tr>
<td>Debugging</td>
<td>3</td>
</tr>
<tr>
<td>Standard report</td>
<td>8</td>
</tr>
<tr>
<td>Report customizable</td>
<td>5</td>
</tr>
<tr>
<td>Exporting report</td>
<td>5</td>
</tr>
<tr>
<td>Framework-Costs</td>
<td>10</td>
</tr>
<tr>
<td>Framework-Structure</td>
<td>7</td>
</tr>
<tr>
<td>Framework -Supported by provider</td>
<td>10</td>
</tr>
<tr>
<td>Framework - Recent version</td>
<td>8</td>
</tr>
<tr>
<td>Framework - Ease of support</td>
<td>8</td>
</tr>
<tr>
<td>Framework - Training</td>
<td>8</td>
</tr>
<tr>
<td>Configuration Effort</td>
<td>10</td>
</tr>
</tbody>
</table>

The steps defined in chapter 5.4.1 are followed to assign the weights to the parameters specified in Requirements document. The decision to assign weights was dependent on the requirement class specified. So if the requirements are mandatory it will be given the highest weight from range 7 to 10 while the recommended requirement will be given the weight from range 4 to 6. The optional requirement are given the weight 1 to 4. The below table gives the Parameters in the requirement specification document with the assigned weights to them:

5.4.3 Giving score to filtered frameworks

In this chapter the discussion is about the score each framework have received after taking the trial of each framework specified in the comparison matrix. The decision to give the score
was discussed with the quality engineers and the engineering team. The score is based on the behavior of each framework it shows with the application under test.

The process was repeated for every framework available for trial from initial filtering as discussed in chapter 5.3.1. The score to the framework is given on the range from 0 to 3. It is based on following reasons:

- If the framework satisfies completely the given parameters it is given the highest score 3.

- If the framework satisfies the requirement of parameters partially it is given the score 1 or 2 based on the performance during trial.

- If the framework does not satisfies the requirement even slightly it is been given the score 0 as it completely dissatisfy the requirement.

The detailed KT decision analysis table is given in Appendix D
5.4. Creation of KT Decision Analysis table

5.4.4 Comparing Results

This section discuss about how the results of individual frameworks was found out. The result of each parameter will sum up at the end to give the total of each framework.

To get the results for a single parameter of a framework, the weight of the parameter is multiplied with the score that parameter received after the trial. The overall result for a single framework can be obtained by summing the results column for a single framework.

Thus, the single framework score will look like this in the below figure 5.4.

The similar process is followed for every framework which underwent trial. There was overall 9 framework which were a part of KT decision analysis table and thus a complete table was created for the selection of framework. Depending on the results obtained there will be a decision taken by us for selecting the framework which best suits the requirement.
Chapter 5. Evaluation of framework

5.5 Framework Selection and Result

This chapter discusses the way selection of framework is done using the KT decision analysis table.

There were many frameworks which matched the requirements to a greater extent but the result of the KT analysis table was in favor of Test Complete. It scored the maximum score in each parameters and lastly topped the table.

There were many frameworks with some extra features like testing devices on cloud like Perfecto mobile but they still could not fulfill the basic mandatory requirements. The test Complete is a commercial license software tool which satisfied most of the requirements completely. Test Complete is also a better solution if it is compared with Ranorex and Appium which scored well in the KT analysis table. These Frameworks are indeed identical in structure, however, Test Complete have some added advantages and performed better in case of iOS or Android operating system.

Also, test complete can execute the scripts from both operationg system iOS or Android from a single machine which gives the user independence to manage two systems. The Framework Ranorex met almost all the requirements, but the tool found difficulty in finding some objects on screen and this was the reason to consider some other option.

The framework Appium matched the requirement specified in chapter 4 completely, there are some discrepancies found while taking the trial. The framework took much effort in writing the test script for a complete system test. This configuration is increased by scripting it twice once for iOS and once for Android. Also generation of reports was one of the lacking feature of Appium which paves the path for a new solution.

Test complete handles these two drawbacks easily and came out to perform better during the trial. The fact that it can be easy to work with developer with programming skills and test engineers with little programming skills makes it a obvious solution for the Selection.
### 5.5. Framework Selection and Result

<table>
<thead>
<tr>
<th>Features Wanted</th>
<th>Weight</th>
<th>Weight</th>
<th>Notes</th>
<th>Score</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported app types-Native</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Supported app types-Hybrid</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Supported app types-Web</td>
<td>2</td>
<td>Yes</td>
<td>3</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Supported OS-Android</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Supported OS-IOS</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Supported OS-Windows Phone</td>
<td>2</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Supported tests-System test (Acceptance test)</td>
<td>8</td>
<td>Yes, Possible with cucumber</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Supported tests-Regression test</td>
<td>7</td>
<td>Yes</td>
<td>3</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Supported tests-GUI testing</td>
<td>5</td>
<td>Yes</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Supported tests-Integration test</td>
<td>3</td>
<td>Yes</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Supported tests-Unit test</td>
<td>2</td>
<td>Yes</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Code-driven testing</td>
<td>5</td>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Linear Scripting</td>
<td>8</td>
<td>Yes</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Data-driven testing</td>
<td>8</td>
<td>Yes, circumstantial with JSON or XML</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Keyword Driven Testing</td>
<td>6</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Modularity-driven testing</td>
<td>5</td>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Style guide testing</td>
<td>2</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Writing test-Script based</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Writing test-UI aided</td>
<td>5</td>
<td>Yes, with UI aided</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Writing test-Record/Playback</td>
<td>8</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Supported interfaces-Eclipse</td>
<td>5</td>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Supported interfaces-Android Studio</td>
<td>5</td>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Supported interfaces-xCode</td>
<td>5</td>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Supported interfaces-Instruments</td>
<td>5</td>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Supported interfaces-Jenkins</td>
<td>5</td>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Script compatibility</td>
<td>9</td>
<td>Yes</td>
<td>3</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Keyword View</td>
<td>8</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Support for Editor</td>
<td>3</td>
<td>Yes</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Access to Third Party app</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Object Detection</td>
<td>7</td>
<td>No, In iOS using Inspector</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Debugger Support</td>
<td>5</td>
<td>No</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Script language</td>
<td>8</td>
<td>Almost all standard languages</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Script execution speed</td>
<td>3</td>
<td>Normal</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Interface versioning</td>
<td>5</td>
<td>Depends on the changes</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Runs on-Local emulator</td>
<td>8</td>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Runs on-Local device</td>
<td>10</td>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Runs on-Emulator in cloud</td>
<td>5</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Runs on-Devices in cloud</td>
<td>5</td>
<td>Yes, with pricing</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Supports code coverage</td>
<td>7</td>
<td>Yes, with Emma</td>
<td>2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Execution in-Parallel Run</td>
<td>5</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Error Handling</td>
<td>5</td>
<td>No</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Defect logging</td>
<td>3</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Standard report</td>
<td>8</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Report customizable</td>
<td>5</td>
<td>Yes</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.4 – The figure represents a framework with its respective score for each parameter
This chapter will discuss about the Implementation of the selected framework for the application under test. Before implementing the selected framework there will be a small discussion about the application under test developed by the company Robert Bosch GmbH. Later on the chapter we can find information on installation of framework, setting the test environment and creating a automated test script from used cases. The test execution and test reporting have been discussed at the end of the implementation phase.

6.1 About the PLR measure&go Application

Bosch Power Tools work on a range of devices from range finders to thermo detectors. This includes all the tools which by means of laser are able to carry out measurements. With the introduction of the PLR 30C these instruments are now fitted with Bluetooth. This measurement can now be transferred to mobile devices and further processed there. For the further processing of these measurements there are mobile Applications which are designed and developed. The Examples include the Apps "PLR measure&go", "GLM measure&document" and "GLM floor plan".

The PLR measure&go application is available to download from the Google Play Store for Android and Apple App store for iOS operating system. Sometimes when user do not have the exact measurements of a fitted cupboard or the user have to measure wall and floor surfaces again and again during renovation work, the application can be used in transferring measurements from the PLR device to the smart phone with the help of Bluetooth.

The PLR measure&go application makes it easier for user to have an access to their measurements from everywhere. The user can prepare information for other users like colleagues, friends and service providers. The benefit of this app is that the user have to take measurement once, and the measured values will be saved for a long period.

The PLR measure&go application is designed such that it can receive measured value from laser range finder device by Robert Bosch GmbH such as the PLR 30C, PLR 40C, PLR 50C.
and other devices which are compatible. This transfer can be done through a Bluetooth. The handling of application is easy, fast and reliable so that there can be no errors due to measurement.

There are following things that can be done with this app:

- The user can make a new project by taking pictures from camera or importing sketches of the plan.

- The user can mark the length and breadth of the photo he takes from camera. Also the user can import exact measured values from the laser range finder device directly.

- Change the color of marked lines to highlight them.

- Enter notes, measured values and angles manually.

- Saves up to 50 unassigned values in a list which user can access for a long-term.

- The user can send entire projects with pictures and measured values as a PDF file just by e-mail.

### 6.2 Framework Installation

Since we work only with Mobile Application, here the discussion is about the installation for Mobile module of Test Complete. The user has to make sure the Mobile module of test Complete was checked during the Setup process for Test Complete. Since Test Complete runs on Windows OS the discussion will be on the prerequisites required for Running test Complete.

#### 6.2.1 For Windows OS

The Test Complete can run both iOS and Android Applications from the Windows machine. The user has to perform preliminary action for getting ready with the environment for test. If the user is testing the Android application following steps are followed –

**Test Complete and Android apps.**

First of all user has to install Java development kit(JDK) and Android SDK.
6.2. Framework Installation

Installing JDK and Android SDK:
When we want to test an Android application, user have to download Java development kit and Android SDK on the test computer. Later these have to be installed correctly.

1. The user can download the JDK from following link:

2. The user can run the downloaded package and follow the instruction in wizard screen.

3. The Android SDK can be downloaded from following link:

4. The user can run the downloaded software and run the installation by following the instructions given in wizard.

5. After installing the JDK and SDK, user have to set up the environment variables.

6. When user finishes installing Android SDK, he can launch Android SDK Manager: Android manager can be found in All Programs –>Android SDK Tools –>SDK Manager.

In the SDK Manager, the user can select the following elements: Android SDK Platform-tools, Android SDK Build-tool and Google USB Driver

7. The user have to install these packages selected from above step. The user has to then update these packages and close the SDK Manager.

Installing the USB drivers for device. If the user want to connect to device then he has to install USB driver on the test computer. Normally these drivers get installed as you plug in the device to test computer.

8. The user has to enable Developer mode option in mobile.

9. To view the connected mobile device on your test computer screen user has to click on Show Mobile screen option in Test Complete window. If the user is able to view the mobile screen in computer then every settings described above are correct.

10. It will be a good practice if the user installs TestComplete Android Agent in mobile phone.

To install it

a. Open the Mobile Screen window.

b. When the user is connected to Test computer and the Test complete is already installed, there is a option in mobile screen to Install Android Agent on the Mobile Screen's toolbar.
Chapter 6. Implementation of Selected framework

TestComplete and iOS apps

The TestComplete can run iOS apps from Windows machine only. The chapter explains how to prepare a testing Environment for iOS apps.

1. Install Apple iTunes

In order to test iOS application, user have to install specified driver on Windows machine. The iTunes can be downloaded from the Apple website. These drivers are part of iTunes package and helps detect the iOS apps on Windows computer. Before user begin to create and execute tests for iOS applications, installing the latest version of iTunes on test computer is important.

2. Check TestComplete Plugins

If the user want to test an iOS application, he has to install the test complete plugins that comes up with the test complete package. Following plugin has to be enabled in testcomplete:

- Mobile Support
- iOS Support.

These plugins are installed and activated during installation wizard automatically.

3. Add “Show Mobile Screen” Toolbar Button

To create iOS tests, user have TestComplete’s built-in Mobile Screen window. When user clicks on Show Mobile screen it can show the connected mobile in its computer screen and can manipulate the things required for test.

6.2.2 With OSX

If the user wants to work on Apple machine and want to install test complete then he has to create a virtual box and specify the operating system as windows system.

The user has to follow all the steps that are discussed under installing test complete on windows.

TestComplete cannot be installed on Mac operating Systems. User has to create a Windows environment to keep working with testComplete on MAC computers.

6.3 Creating a test script from Use cases

6.3.1 Defining Use Case

According to the test Plan from PLR measure&go, we take out a scenario where the user navigates to different screen on mobile window. The steps are described below:
1. Launch the application.

2. User is able to see the splash screen being launched.

3. Initially user clicks on the cross button at the navigation bar and goes to Create new project Screen.

4. The user clicks on Add new project icon and go to the next screen for filling details of Project.

5. The user fill all the details in the respective fields and add a picture to the project.

6. User clicks on Create project button and the project is created if atleast the Project name was entered.

7. User navigates to created Project screen and click on work space icon to move to Create a plan for that project.

8. User opens a graph paper and draw different sketches and add text boxes in graph.

9. User goes back to project screen and check settings screen.

10. At last user goes to delete the created project and long press on delete button will going to delete it.

### 6.3.2 Test Script for Android Apps

User can select to create a test script using three techniques.

1. Recording the steps of test.

2. Manually adding the objects from screen and creating the keyword tests.

3. Writing a code for the objects in scripting language.

In Test Complete, mobile tests are either recorded or created step by step manually in form of keyword tests or scripts. It can be a bit easier to record the test initially and then update and enhance the recording of test.

When user record a test, he interact with the tested Android application as an final-user would navigate through the mobile application screen. This may include filling out forms and so on. TestComplete captures all actions a user would perform in the application and adds them to the test.

### Adding Objects to Project: By Recording

a) To start recording, do any of the following:

- Click Record Keyword Test or Record Script on the Test Engine toolbar.
• Select Test and Record and select Record Keyword Test or user can select Test and Record and Record Script from TestComplete’s main menu.

![Figure 6.1 – The figure shows the option for Recording the Script](image)

b) Perform test actions in the Mobile Screen window as specified in Use case discussed in chapter 6.3.1
6.3. Creating a test script from Use cases

**Adding Objects to Project: By Keyword test**

User can add each step by step script by using Keyword tests. User has to select Keyword tests in Project folder and click on Add New items under project folder and click on new keyword test.

If the application was prepared for testing, and it has TestComplete Android Agent installed on the device then the hierarchy of all the objects with their properties can be viewed easily. This can be checked using the object spy window. The user can choose and explore any object for testing from the mobile window in any of the following ways:

**Making Keyword Tests**

1. The user can simply right click on the Keyword test window and Insert New Operation and select to 'add device'.

2. The user can add next step by again clicking on Insert New operation and Select 'Run tested Apps' and selecting the App under test from pop up window

3. User can click on Insert New operation -> Insert On screen action and a pop up opens to select object.

To specify the object with the target glyph

- Click on the target glyph and keeping the mouse button long pressed, point to the object as if user is dragging the icon. In this process the TestComplete's window minimizes.

- The user can then drag the icon to the desired object on mobile window. He can also control and release the mouse button on the application object

TestComplete will going to display the complete name of the selected object in the Object spy window. If user has enabled automatic name mapping, it will be the mapped name.

The user can select the object on mobile screen to point and the address of the object will be listed in the Objects address. User can check the address is correct by clicking on Highlight, and the selected object should be highlighted.

4. The user can select next and there user can specify Method or property so that selected action can be performed on object. Eg: User can select Touch Action so the required object will be touched.

5. At last screen, user can click finish and a new keyword step is being formed.

6. Repeat this process for all the test steps from Test Plan and you will be able to create a complete keyword Test script looking like in figure 6.4.
Chapter 6. Implementation of Selected framework

Figure 6.2 – The figure represents adding object from screen using Keyword tests

Figure 6.3 – The figure represents adding action to the keyword test

Adding Objects to Project: By Scripting

A script is a sequence or function in one of the scripting language which is supported. A script can be grouped into unit case script and every project may need as many unit script as required. In the Object browser window, a user can view all the hierarchy of project elements and the units that belong to project script item. When the user creates a new project a new script with a unit test is already created automatically.

Writing the script using Code Editor is done by:

1. Selecting the language to script.
6.3. Creating a test script from Use cases

2. The user can follow the test plan and start getting the properties of the individual object by going to object browser and locating the object in hierarchy of the application. The Object Browser shows all the applications which are running. User can then view the hierarchy of objects in applications and select desired object properties and methods. To go to the Object Browser user has to just click Object browser tab or select View and Object Browser from the main menu. The Object browser is displayed in figure 6.5.

Figure 6.4 – The figure represents each step from keyword view for the Use cases

Figure 6.5 – The properties from the Object Browser Window
3. The user can copy the property of the object and use it in the Script editor to specify the object and later appending the desired action on it (in figure 6.5).

4. The Script will look like this (in figure 6.6) if user adds on each step:

![Image of Visual Script interface]

Figure 6.6 – The figure shows the code written for the use cases in Visual Script
6.4 Execution Of Test Scripts

The user should make sure of few things before Running Automated Tests.

1. The user should run automated tests in an environment with correct pre conditions. This may be useful as it will avoid possible errors due to environment inconsistencies.

2. If a test runs the application, user has to verify if the application is not running already.

3. If a test works with an already running application, make sure the application is in the right state for the test.

4. If a test creates files, make sure these files do not already exist.

Running Tests from Keyword test

If the user selects to run the entire project or a project suite

• The user has to make sure that the Test run is already configured. This can be done on the Test Items page of the project editor or project suite editor.

• The user can then Select to Run Project or Run complete Project Suite from the Test menu items.

– or –

The user can Select Run and Run Project or Run and Run Project Suite from the Test Engine toolbar.

To run a keyword test

• The user has to right-click on the required script unit in the Project Explorer window and select Run from the context menu(in figure 6.7).
Chapter 6. Implementation of Selected framework

To run a script routine

- Right-click the needed script unit in the Project Explorer and select Run | Run Routine Name from the context menu.

  ![Figure 6.8](image)

  Figure 6.8 – The figure shows the option for running a test from the project view

- or –

  The user can go in the Code Editor window and place an insertion point inside the routine code. If the user selects to click Run Current Routine from code editor toolbar then the script can be executed.
6.5. Test Results and Reporting

After running a test, user can view test results in Testlog window. The test log view contains many panels. The left panel are Log Items which show a tree-like structure of the results from test. The pages on the right of panel shows the results contained in the selected log item. The layout of pages that display the output results solely depends on the type of test. The very common kind of layout is the Generic Test Log. This kind of layout shows the output from keyword tests, script based tests, unit case testing and more tests.

![Test Results and Reporting](image)

Figure 6.9 – The figure represents the successful test for a implemented Use case

**The log item**

The Log Items subpanel of the Test Log panel shows a hierarchy like structure of test results. Test results are managed by name of projects, test items and underlying tests related to it. They can be keyword tests, script routines etc. The log item names demonstrate the project elements that were executed. When a user select an item from the log tree, the panel in the right side shows the results that are contained in that item. The following image shows a sample log structure:

The icons of the log tree items indicate the test status:

- The green tick resembles the test have completed successfully.
- The red cross indicates the test is completed with errors and is referred as failed. All the parent items from log is also considered fail.
Chapter 6. Implementation of Selected framework

- The yellow exclamation indicates the item is completed with warnings. The parent items of such an item are understood as completed with warnings.

If the user wants to view the test result inside a particular log item the he has to select that particular item in the log tree. TestComplete can show the results in the panel located on the right side.

Test Reporting

The framework testcomplete saves results of each test execution in the XML format and includes any image files in the log. The results are kept in the Log subfolder of project’s folder.

TestComplete’s test log shows the project suite result of execution or a project run selected in the Project Explorer panel. For example user can send the result files to his peers, who does not use TestComplete, and see the results there.
Viewing Results in Web Browser

The user can view test results in web browser, he should export results from the test log. You can do this by using the context menu items.

The user can also view the exported results in web browser. For doing this, open the desired test log in the Workspace panel and right-click on the needed log item from the hierarchy displayed. The user can then choose to view Results in his selected browser from the context menu.

Test results can be exported in one of the following formats:

• **Unpacked Storage** - A number of .xml, .htm, .js and image files that are saved in the folder user have specified. The “main” file is index.htm. In order to see exported results, open the main file in any browser window.

• **Multipart Hypertext Storage** - A single .mht file that can be viewed in web browser. When user selects to export the result generated in a preview, the tool will create a temporary .mht file and show it in web browser.

Irrespective of the format user chooses, the exported results will have the same outlook in the web browser window:

![Figure 6.10 – The figure shows the successful report opened in a browser](image)

The results displayed in web browser is just like the Test Log window. The left part of the
Chapter 6. Implementation of Selected framework

browser window is occupied with the similar test item log tree that is displayed on the left of the Test Log window. On the right side of the tree the browser window shows the details of the log item chosen in that tree. Based on the log item type like Project, Project Suite or Script, the browser window can display more panels below or to the left side of the log item contents. For example, if the chosen log item corresponds to a project, the browser window will show the Message panel below the log contents. If user is browsing the script execution results, the window will show the extended log messages to the right hand side of the “main” log contents. The extra information panels and the test log tree can be chosen to be hidden. This can be done by clicking the arrow button in the panel caption.
7 Conclusion

This chapter discuss about the summary of the entire work and all the important results gathered during the process. There will also be a prospectus given for the task that can be done for the future research.

7.1 Results

By introducing test automation in System testing of project life cycle, the department PT-MT/QMM will be able to test the mobile application in future more accurately and efficiently. In fact, the testing team can test more number of test with a far better coverage then a manual test are concerned.

We are also able to conclude at this moment that the Effort estimation we did in chapter 3.2 was correct and the actual result showed far better responses then assumption. The total test effort for Manual test for first iteration of testing was 34 men hours while the automation test effort for one Use case took around 48 men hour. This is very close to our assumption of 54 men hour required for initial test Automation. The further iteration of test Automation will require less amount of effort and so the assumption of incrementing the effort by 5 men hour can be justified.

The actual graph of a complete release of mobile application can be shown when the framework is implemented in actual System testing process. The test Engineers working on the framework will be able to get the real data for effort utilized for test automation in different releases of mobile application.

After extensive evaluation and research the selection of the frameworks was in favor of TestComplete which is a commercial licensed tool.
Chapter 7. Conclusion

7.2 Summary

The department PT-MT/ QMM of company Robert Bosch GmbH wanted to find a framework for Automation testing of Mobile Application in System testing process.

For this a vast amount of interviews and brainstorming sessions were held with the developer and the test engineers for knowing the actual requirement. Then a Requirement specification document was created as a base for the further research. The list of frameworks available in market for Automation testing was collected and were compared in a matrix which contained the parameters from requirement specification document on Rows and the list of frameworks in the column. The details of the framework was taken out from internet, framework websites and information from previous researches.

The framework were filtered from the comparison matrix depending on the priority of the parameters specified in Requirement Specification document. If the framework supported all the mandatory requirements then the framework was considered for further level of research. At the next point, a low level research was conducted where the filtered frameworks were taken for a trial and the application under test was checked for the behavior it shows.

After taking the trial, the decision to select a framework was done using a KT Analysis table. The parameters from requirements specification were assigned weights and each framework was given a score based on their performance during the trial. The result of a individual framework was a product of weight and score for each parameter. Thus a overall score was calculated by summing up all the results. The framework with the highest overall result is the preferred framework for selection. TestComplete comes out to be the clear selection for the framework to be implemented in System test cycle.

The implementation of TestComplete was done for a use case from System tests. TestComplete was installed on a test computer and all the testing environment was set up. A complete documentation for installing the tool and creating scripts have been documented and shared with the department. The mobile test was created for ios and Android operating system and the execution was performed for a single cycle of System test. The results have been reported in the form of a test Report which can be exported to other users using email options. The complete documentation was prepared for the further use of TestComplete in the test process of PT-MT/QMM department.
7.3 Further Outlook

The Test tool for Automation testing has been selected and implemented for System test in PT-MT/QMM. It will improve the testing capabilities of a tester and will improve the overall process to be more accurate and efficient.

There are still some areas that can be explored during the further research for coming months. The tool can be used to build scripts which can be a part of a common framework of testing. This can be done by designing a common framework which contains reusable libraries and scripts such that these libraries can be used directly in any projects which want to use the TestComplete tool and implement automation testing. Obviously the test engineers working on it will be in a better position to work on such implementation as they will gain some experience working with this tool.

This will further increase the advantage of a tool that can support Desktop, Web and mobile applications as a key feature. There can be a work required to see how TestComplete can support testing at a lower level of test cycle like Unit testing. The department can then utilize a single tool to do the testing from initial test phase to final test phase like user acceptance testing.
Appendix A

Requirements specification

Aim: To Find a Framework for Automation testing of Android and IOS mobile apps for PT-MT/QMM.

Present Scenario: The testing Team uses Manual techniques to test the Apps on different OS platform and on various devices. The process is time consuming and the work is repetitive.

Proposed Plan: The PT-MT/QMM proposes to introduce an automated framework for System Testing which will be able to write a script once and it can be reused for further Testing of all other apps under construction. This will Enhance quality and decrease time for testing.

Intended Audience

This document mainly target on the following stakeholders

1. PT-MT/QMM
2. PT-MT/ELF

This document consists of several subsections in which we have described the software requirements for our quality assurance automation system.

Requirement are categorized under three types:

• Mandatory Requirement (As Must to Have)
• Recommended Requirement(As Good to Have)
• Optional requirement(As Nice to Have)

Stakeholders whom we mention above can get a brief idea about the system by reading this document. In this document we have briefly described features of the system functionality we are expecting from the system and how this system going to help for the above stakeholders.
Appendix A. Appendix A

Assumption:

It is assumed that the reader has some basic understanding of Software Testing. The keywords used may be the technical terms used by the developer or Tester.

Requirement

• Support for Various Applications and Platforms

Support Programming tools (Recommended Requirement)

o Eclipse

o Xcode

o Android Studio

• Version control tool:

Tortoise SVN / Subclipse / svnX as (Recommended Requirement)

• Continuous Integration tool: (Recommended Requirement)

o Jenkins

• Support for Operating Systems (Mandatory Requirement):

It's not necessary that we find a framework which supports both OS Android and iOS. But it's good to have Single framework which supports both OS.

• Support for Mobile App Types

The framework should support

o Native apps (Mandatory Requirement): These apps live on the device and are accessed through icons on the device home screen. Native apps are installed through an application store (such as Google Play or Apple's App Store).

o Web Apps (Optional Requirement): Web mobile applications are software programs that run directly from the web browser on mobile phones and tablets. These web-based mobile apps do not get installed on your handheld mobile device and are run on web-hosted servers

o Hybrid App (Mandatory Requirement): Hybrid apps are part native apps, part web apps. Like native apps, they live in an app store and can take advantage of the many device features available.

• Support for Various Software Testing Types

The Framework has to support:
System testing as Mandatory Requirement

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

Regression testing as Recommended Requirement

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn’t resulted in another functionality or business rule violation.

Acceptance Testing as Mandatory Requirement

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

In our Test project the System test and Acceptance test are combined and hence the app is being tested against the Acceptance Criteria under full system implementation.

Integration testing as Optional Requirement

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

Unit testing as Optional Requirement: Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing.

Graphical user interface(GUI) testing (as Recommended Requirement)

GUI testing is the process of ensuring proper functionality of the graphical user interface (GUI) for a given application and making sure it conforms to its written specifications.

In addition to functionality, GUI testing evaluates design elements such as layout, colors, fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links and content.

Creating and Driving Automated Test Case

Linear Scripting as Mandatory Requirement

This is the simplest of all frameworks, and very similar to the Record-Playback mechanism – where every single interaction is captured and detailed out in the script/feature file. A linear script is what you ends up with when you record a test case manually.

Code-driven testing as Recommended Requirement
Appendix A. Appendix A

Code driven test automation is a key feature of agile software development, where it is known as Test-driven development (TDD). Unit tests are written to define the functionality before the code is written. Only when all tests pass is the code considered complete. In our case that can be after the source code is written.

Data-Driven Testing as Mandatory Requirement:

A Data-Driven testing framework refers to the repeating of the same test case scenario using different data sets – making the test case, data centric. A Data-Driven script is parameterized, replacing hardcoded values with variables, in which the data may be loaded from a number of sources, including: the Gherkin file, text files, Excel files, CSV files and databases.

Keyword driven testing as Recommended Requirement

It is an approach in which the control to check and execute Test cases is maintained with external data files. So the test data and the operations/sequence of the test is planned in external data file and extra library is need to interpret this data in addition to the conventional script. It is an extension of data driven testing.

Modularity-driven testing as Recommended Requirement

This means that it is possible to the small unit test scripts can be used to construct larger test cases.

Style guide conformance testing of the layouts as Optional Requirement

This means that it’s possible to write a test script which checks the app layouts on their style guide conformation. The style guides can be from Robert Bosch GmbH, Apple or Google.

Automated Test Scripting

Manual Scripting as Mandatory Requirement

It means that the Tester can write a script in a standard language using standard languages, such as VBScript or JScript rather than their dialects. This will decrease the time needed to get acquainted with a product.

UI aided scripting as Optional Requirement

UI aided scripting means that the tester can use a UI to develop a script. E.g. he can drag and drop predefined commands and customize them if needed. The script is automatically written by the framework based in the commands so that the tester doesn't have to write a single line of code.

Record/ Playback function as Mandatory Requirement

Record/ Playback function means that the tester is able to record a test case by executing the App on a device or emulator. The script is automatically written by the framework based in the commands so that the tester doesn't have to write a single line of code.
• Scripting Support

o Single Script compatibility as Mandatory Requirement

It is very preferable that the scripts created for iOS and Android are compatible / similar. It is recommended that a single script is used for executing on both iOS and Android devices without much change.

o Keyword Script View as Mandatory Requirement

Preferably the Script View is keyword-driven. That means that there are abstraction layers so that also non-programmers can understand the scripts by looking at it. So there must be two views- one for Programming language specific understanding like Jscript or VB Script and one view for non-programmers.

o Support for editor (as Optional Requirement)

The editor supports features that simplify and speed up script writing: syntax highlighting, outlining, code completion, bookmarks, Ctrl-click navigation between script routines and many others.

o Object Identification (as Recommended Requirement)

The tool should have a powerful object identification capability. An Object is any entity which can be located in the app. It can be a button, dropdown menu or a cross for closing the app.

Support for all object identification methods: more specifically, the tool should be able to identify an object by its Native ID, Image Recognition, Text Recognition and Web HTML5 (DOM). This is crucial otherwise the maintenance of the tests will be very poor.

o Debugger Support (as Recommended Requirement)

The debugging (Removing error) functionalities, such as breakpoints eases the work of Tester by locating the point of failure. Using these features you can easily find and correct bugs or find workarounds.

o Script Execution Speed (as Optional Requirement)

The script speed execution should be in an adequate proportion to the script size. For example a big modularity-driven test case executed on several devices should not take more than one day.

o Interface versioning (Scripts working on new versions): (as Recommended Requirement)

The scripts should be able to work on newer version of the tool or any updates of them. There should be a special attention paid that there is no changes made on the existing scripts if the new update of tool is available.
Appendix A.

• Test Execution

  o Test Run on Devices (as Mandatory Requirement)

  The framework has to support the test execution on at least a local emulator or a local device. Otherwise an emulator in cloud or preferably it gives us the opportunity to use several devices accessible over the cloud.

  o Multiple/Parallel test Run (as Recommended Requirement)

  There must be possibility of testing device in parallel from one machine. We should be able to create our test once and then play it back from one machine on all the available devices at one time.

  o Test/Code coverage: (as Recommended Requirement)

  Code coverage is used to describe the degree to which the source code of a program is tested by a particular test suite. A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage.

  It's important that we have an indication of code coverage. Therefore the framework should at least support one existing code coverage tool for eclipse and Xcode. Common tools for Eclipse are JaCoCo, CodeCover, and Cobertura. For Xcode it is GVOC. Otherwise it should provide an own solution.

  o Defect Logging : (as Optional Requirement)

  It's important that the framework should report failed results and they can be logged in the present bug tracking tool. So, there should be a connectivity between the automation tool and Bug tracking tool.

• Test Reporting

  o Standard report Format: (as Mandatory Requirement) The format of standard report should be known and predecided as per data format. The report format should be detailed view with possibility of screenshot of the steps executed. Also, in case of fail results the failed steps can be located.

  o Report customizable: (as Recommended Requirement) Test Reports can be customized so that they are also available in a user friendly form. Customized means the way of the representation of the data which is generated during a test run.

  o Exporting Reports: (as Recommended Requirement)

  An automated testing tool should give users the opportunity to export test results to external files. For instance, it may be useful, when you need to view test results on other computers that do not have this testing tool installed. Also, you should check whether the testing tool can
export the test log so that you can send the results to your boss or colleagues.

- Configuration effort: (as Mandatory Requirement)

The configuration of the framework should be connected with an acceptable effort. This parameter will be more decisive in the end of the framework selection.

- Pricing Policy

  - Costs: (as Mandatory Requirement)

    The framework should be preferably open source, but if the requirements are more fit by a framework with costs there is budget for paying the license. If paying for a license there should be a comparison for which license fits the need. For e.g. we need license for one person or a floating license for 5 user when logging in at one time. The validity of license required is for one year or license for flexible use when required. Also it has to be checked if the license are depending on per devices used or valid for all device and paid once a year.

  - Structure: (as Recommended Requirement) Everything is accepted as long as the required interfaces are supported.

- Maintenance:

  - Assured by provider: (as Mandatory Requirement)

    The Maintenance should be assured by the provider especially if we have to pay for a license. In that case it is a must have that we have no big adjustment needed on our side.

  - Recent version: (as Mandatory Requirement)

    If the framework is open source the last update has to be at least in 2014. Preferably it is in the actually month.

  - Ease of support: (as Mandatory Requirement)

    There have to be a support available to make sure that we can solve upcoming problems with the framework. For that there should be at least enough online learning material like tutorials etc. or a community. Except we pay for a license than there has to be support by the provider.

- Free Trial Versions (as Mandatory Requirement)

  An important factor is the availability of a trial version. Even if a product seems to meet our requirements, the best way to determine that is to download and use it before purchasing. Also, there should be a possibility for a demo of the features and functions of Test tool from the company on specifically Bosch apps.
Below is the Comparison Matrix document:
### Appendix A. Appendix B

#### Figure A.1 – Comparison Matrix with List of Framework Part-1

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Criteria</th>
<th>Test Tools</th>
<th>Robot</th>
<th>Appium</th>
<th>Catechah</th>
<th>Concurrency</th>
<th>eggPlant Mobile</th>
<th>Expresso (android test)</th>
<th>Flippy (aristocratic test)</th>
<th>HP Mobile Center</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratings</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported apps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported O/S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smoke testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Execution and Driving Automated Test Case |          |            |       |        |           |             |                  |                        |                        |                  |
| Long-running |          |            |       |        |           |             |                  |                        |                        |                  |
| Load testing |          |            |       |        |           |             |                  |                        |                        |                  |
| Stress testing |          |            |       |        |           |             |                  |                        |                        |                  |
| Regression testing |          |            |       |        |           |             |                  |                        |                        |                  |
|           |          |            |       |        |           |             |                  |                        |                        |                  |
| Automated test scripting |          |            |       |        |           |             |                  |                        |                        |                  |
| Script based |          |            |       |        |           |             |                  |                        |                        |                  |
|           |          |            |       |        |           |             |                  |                        |                        |                  |
| UI aspect |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Treatment interface |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Rendering |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Script execution speed |          |            |       |        |           |             |                  |                        |                        |                  |
| Interface rendering |          |            |       |        |           |             |                  |                        |                        |                  |

| Test execution |          |            |       |        |           |             |                  |                        |                        |                  |
| Local test |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Remote test |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Support code coverage |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Parallel test |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Performance test |          |            |       |        |           |             |                  |                        |                        |                  |

| Configuration effect |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| License validity |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Elasticity |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Error Handling |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Data logging |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |
| Exporting report |          |            |       |        |           |             |                  |                        |                        |                  |
|          |          |            |       |        |           |             |                  |                        |                        |                  |

| Performance |          |            |       |        |           |             |                  |                        |                        |                  |
|            |          |            |       |        |           |             |                  |                        |                        |                  |

| Memory |            |            |            |            | per year             | per year             |            |            |                                   |
|          |            |            |            |            |                           |                           |            |            |                                   |
| Structure | Local Software | Open Source | Architecture Library Library Local Software Library Java/JDK, UML compatible Java/JDK, UML compatible Java/JDK, UML compatible Java/JDK, UML compatible |
|          |            |            |            |            |                           |                           |            |            |                                   |
| Maintenance |          |            |            |            |                           |                           |            |            |                                   |
|           |          |            |            |            |                           |                           |            |            |                                   |
| File System |          |            |            |            |                           |                           |            |            |                                   |
|           |          |            |            |            |                           |                           |            |            |                                   |
| Support Files |          |            |            |            |                           |                           |            |            |                                   |

| Other Tools |          |            |       |        |           |             |                  |                        |                        |                  |
|            |          |            |       |        |           |             |                  |                        |                        |                  |
|            |          |            |       |        |           |             |                  |                        |                        |                  |

80
Figure A.2 – Comparison Matrix with List of Framework Part-2
Appendix C

Following is the reference for the test framework

**Abbot**


**Appium**

https://saucelabs.com/mobile/android-testing/android-test-automation

https://www.linkedin.com/grp/post/86204-5945515983136395268

http://appium.io/slate/en/v1.0.0/?java#android-coverage

https://discuss.appium.io/t/how-to-get-the-pass-fail-report-for-appium-on-android-mobile-application/2946/6

https://blog.codecentric.de/en/2014/05/android-ui-testing-appium/


https://discuss.appium.io/t/appium-tests-via-android-studio-and-gradle/1358

**Calabash**

http://calaba.sh/


https://groups.google.com/forum/#msg/calabash-ios/sdP1K0o25bl/-pfBnsq3G28J


https://github.com/calabash/calabash-ios
Appendix A. Appendix C

http://de.slideshare.net/NielsFrydenholm/automated-ui-test-on-mobile-with-cucumber-calabash
https://groups.google.com/forum/#topic/calabash-android/pZzWY5XUL6A

Concordion

http://concordion.org/
http://concordion.org/Tutorial.html
https://wiki.jenkins-ci.org/display/JENKINS/Concordion+Presenter+Plugin

eggPlant Mobile

http://www.testplant.com/eggplant/testing-tools/eggplant-integration/
http://www.testplant.com/demos-and-case-studies/testing-use-cases/
http://www.testplant.com/eggplant/testing-tools/eggplant-mobile-eggon/
http://docs.testplant.com/?q=eggPlant-Manager/exporting-results-and-creating-reports
http://www.testplant.com/eggplant/testing-tools/eggplant-manager/

Espresso

https://code.google.com/p/android-test-kit/wiki/EspressoSetupInstructions
https://code.google.com/p/android-test-kit/
https://www.youtube.com/watch?v=qtKx1WxK7cw
http://googletesting.blogspot.de/2013/10/espresso-for-android-is-here.html

Fitnesse

https://www.youtube.com/watch?v=j-hTZmrVGbI
https://wiki.jenkins-ci.org/display/JENKINS/Fitnesse+Plugin
https://dzone.com/refcardz/getting-started-fitnesse

Instruments

http://stackoverflow.com/questions/23210064/xcode-data-driven-testing

iOS-driver
http://de.slideshare.net/davidodowd/ios-driver-presentation-copy
http://ios-driver.github.io/ios-driver/?page=home#download

JUnit
http://junit.org/
https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin
http://junitpdfreport.sourceforge.net/managedcontent/
http://www.soapui.org/reporting/generating-html-reports.html
https://code.google.com/p/junitum/
https://github.com/junit-team/junit/wiki/Categories

Keynote
http://www.keynote.com/solutions/testing/mobile-testing
https://www.keynote.com/resources/white-papers/testing-strategies-tactics-for-mobile-applications

M-eux
http://www.jamosolutions.com/shop/

MonkeyTalk
https://www.cloudmonkeymobile.com/monkeytalk

M3
Appendix A. Appendix C

Perfecto Mobile
http://www.perfectomobile.com/
https://www.youtube.com/watch?v=yMlNL8T49js
http://help.perfectomobile.com/
https://www.perfectomobile.com/Solutions/DevOps
http://www.perfectomobile.com/solution/test-automation

Ranorex
http://www.ranorex.com/blog/automated-regression-testing-with-ranorex
http://www.ranorex.com/
http://www.ranorex.com/automated-gui-testing-tools.html
http://www.ranorex.com/forum/is-ranorex-is-used-to-generate-code-coverage-t662.html
http://www.ranorex.com/support/user-guide-20/lesson-8-reporting.html

Robotium/ Robotium Recorder
https://code.google.com/p/robotium/w/list
https://code.google.com/p/robotium/
http://stackoverflow.com/questions/4850641/how-to-generate-test-result-report-using-robotium
http://robotium.com/pages/installation-android-studio
http://robotium.com/products/robotium-recorder
http://robotium.com/pages/user-guide

SeeTestAutomation
https://experitest.com/automation/

Selendroid
http://selendroid.io/
http://selendroid.io/native.html
http://selendroid.io/hybrid.html
https://wiki.jenkins-ci.org/display/JENKINS/AppThwack+Plugin
http://www.seleniumhq.org/download/

**SOASTA TouchTest**

http://www.soasta.com/touchtest/

**TestObjects**

https://testobject.com/features
https://help.testobject.com/docs/testing-tools/manual-testing/

**TestFlight**

http://www.androidcentral.com/testflight-beta-testing-platform-killing-android-support-mar-21

**HP Mobile Center**

https://www.youtube.com/watch?v=ASLznBvS_jw

**Silk Mobile Test**


**Test Complete:**

https://smartbear.com/product/testcomplete/overview/
https://support.smartbear.com/viewarticle/75299/?st=0
https://en.wikipedia.org/wiki/TestComplete
Appendix D

Below is the complete KT Decision Analysis Table used to Select the Framework
Figure A.1 – KT Decision Analysis Part I
## KT Decision Analysis Part II

<table>
<thead>
<tr>
<th>SeeTest</th>
<th>Ranorex</th>
<th>Perfecto Mobile MobileCloud</th>
<th>Test Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notes</strong></td>
<td><strong>Score</strong></td>
<td><strong>Result</strong></td>
<td><strong>Notes</strong></td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>21</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>1</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>18</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>1</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>1</td>
<td>9</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>1</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>14</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>Vb Script</td>
<td>1</td>
<td>8</td>
<td>VB.NET, C#, IronPython</td>
</tr>
<tr>
<td>Normal</td>
<td>3</td>
<td>9</td>
<td>Normal</td>
</tr>
<tr>
<td>Depends on the changes</td>
<td>1</td>
<td>5</td>
<td>Object identification</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>15</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>14</td>
<td>Yes, using Jenkins plugin</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>1</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Licence after trial</td>
<td>1</td>
<td>10</td>
<td>Floating licence</td>
</tr>
<tr>
<td>On Sever accessed by a Browser</td>
<td>3</td>
<td>21</td>
<td>Local Software</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>30</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>Yes, on 08/2015</td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>24</td>
<td>High</td>
</tr>
<tr>
<td>Online material Webhelp</td>
<td>2</td>
<td>16</td>
<td>Online Material + Ranorex team</td>
</tr>
<tr>
<td>Medium</td>
<td>3</td>
<td>30</td>
<td>Low</td>
</tr>
</tbody>
</table>

| 101 | 712 | 98 | 707 | 89 | 619 | 106 | 746 |
A.1 Android Code

The code for System test for Automation of PLR measure&go Android application is as below:

Sub Test14()

'Specifies the mobile device as current for the test commands working with mobile devices.
Call Mobile.SetCurrent("D2005")

'Runs the "PLR measure and go" tested application.
Call TestedApps.PLR_measure_and_go.Run

'Touches the mobile device's screen at the specified coordinates.
Call ImageRepository.diy_plr.ImageView_UnnamedCtrl.Touch(347, 396)

'Delays the test execution for the specified time period.
Call Delay(5000)

'Simulates a touch event at the specified point.
Call Mobile.Device.Touch(157, 513)

'Delays the test execution for the specified time period.
Call Delay(1000)

'Touches the mobile device's screen at the specified coordinates.
Call ImageRepository.diy_plr.ImageButton_UnnamedCtrl.Touch

'Simulates a touch on the View control at the specified position in control-relative coordinates.
Appendix A. Appendix E

If no coordinates are specified, a touch is simulated in the center of the control.

_create_new_project.Touch

'Enters text in the edit control.

Call Aliases.Device.Process_diy_plr.RootLayout.Layout_NO_ID.scrollView_NO_ID.
EditText_project_name.SetText("Completetest1")

'Enters text in the edit control.

Call Aliases.Device.Process_diy_plr.RootLayout.Layout_NO_ID.scrollView_NO_ID.
EditText_address_street.SetText("Leinfelden")

'Simulates a touch on the View control at the specified position in control-relative coordinates.
If no coordinates are specified, a touch is simulated in the center of the control.

Call Aliases.Device.Process_diy_plr.RootLayout.Layout_NO_ID.scrollView_NO_ID.
Layout_img_project_container.imageView_add_proj_photo.Touch

'Delays the test execution for the specified time period.

Call Delay(5000)

'Touches the mobile device's screen at the specified coordinates.

Call ImageRepository.android.TextView_Camera.Touch

'Delays the test execution for the specified time period.

Call Delay(5000)

'Simulates pressing a physical (on-device) button.

Call Mobile.Device.PressButton(mbkVolumeDown)

'Delays the test execution for the specified time period.

Call Delay(5000)

'Simulates a touch event at the specified point.

Call Mobile.Device.Touch(126, 778)

'Performs a single touch on the specified button.

Call Aliases.Device.Process_diy_plr.RootLayout.Layout_NO_ID.scrollView_NO_ID.Button
_Create.TouchButton
A.1. Android Code

`Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

CallAliases.Device.Process_diy_plr.RootLayout.Layout_main_project_layout.GridView_project_grid_view.Layout_NO_ID2.Layout_NO_ID.ImageView_project_item_image.Click(0, 0)

`Simulates a touch on the View control at the specified position in control-relative coordinates. If no coordinates are specified, a touch is simulated in the center of the control.


`Simulates a touch on the View control at the specified position in control-relative coordinates. If no coordinates are specified, a touch is simulated in the center of the control.


`Touches the mobile device's screen at the specified coordinates.

CallImageRepository.android.TextView_Graph_pape.Touch

`Touches the mobile device's screen at the specified coordinates.

CallImageRepository.diy_plr.ImageView_Drawline.Touch

`Simulates a touch event at the specified point.

CallMobile.Device.Touch(50, 100)

`Plays the Gesture2 [Default] gesture on the current device.

CallAndroidGestureCollection.Default.Gesture2.Execute

`Touches the mobile device's screen at the specified coordinates.

CallImageRepository.diy_plr.Edt_measurement.Touch

`Enters text in the edit control.


`Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.


`Touches the mobile device's screen at the specified coordinates.
Call ImageRepository.diy_plr.Edt_text.Touch

'Simulates a touch event at the specified point.

Call Mobile.Device.Touch(250, 200)

'Enters text in the edit control.

EditText_text_edit_note.SetText("This is awesome")

'Touches the mobile device's screen at the specified coordinates.

Call ImageRepository.diy_plr.Kb_done.Touch

'Touches the mobile device's screen at the specified coordinates.

Call ImageRepository.diy_plr.Edt_StickyNote.Touch

'Simulates a touch event at the specified point.

Call Mobile.Device.Touch(300, 400)

'Enters text in the edit control.

EditText_edittxt_sticky_note.SetText("This is a sticky note")

'Performs a single touch on the specified button.

Layout_header_layout/Button_Done.TouchButton

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

_measurementList.ImageView_measurementlistHandle.Click

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

View_navigationBar.Layout_navigationbarHandle.ImageView_handleImage.Click

'Plays the Gesture1 [Default] gesture on the current device.

Call AndroidGestureCollection.Default.Gesture1.Execute
A.1. Android Code

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

_navigationBarLayout_navigationbarContent.Layout_project_sketch_header.Layout_layout
_back.TextView_Back.Click

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.'

_email.Click

'Touches the mobile device’s screen at the specified coordinates.

Call ImageRepository.android.FrameLayout_Mail_once.Touch

'Simulates keystrokes on the device.’

Call Mobile.Device.Keys("fixed-term.ankit.shrivatri@de.bosch.com")

'Touches the mobile device’s screen at the specified coordinates.

Call ImageRepository.android.SendMail.Touch

'Simulates pressing a physical (on-device) button.

Call Mobile.Device.PressButton(mbkBack)

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

_layout_back.TextView_Back.Click

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

_settings.Click

'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.

_layout_cancel.TextView_Back.Click

'Simulates a click on the View control at the specified position using control-relative coordi-
nates. If no coordinates are specified, a click is simulated in the center of the control.


'Simulates a long touch on the View control at the specified position in control-relative coordinates. If no coordinates are specified, a long touch is simulated in the center of the control.


'Simulates a click on the View control at the specified position using control-relative coordinates. If no coordinates are specified, a click is simulated in the center of the control.


'Simulates a touch on the View control at the specified position in control-relative coordinates. If no coordinates are specified, a touch is simulated in the center of the control.


A.2 IOS Code

The similar test script for a similar USED case in iOS is:

```plaintext
function Test2()

//Specifies the mobile device as current for the test commands working with mobile devices.
Mobile.SetCurrent("iPhone 6");

//Runs the "PLRMeasure_Gocopy" tested application.
TestedApps.PLRMeasure_Gocopy.Run();

//Specifies the mobile device as current for the test commands working with mobile devices.
Mobile.SetCurrent("iPhone 6");

//Performs a touch on the 'OK' button of the 'alertview0' alert view.
Aliases.Device.processPlrMeasureGo.window2.alertview0.TouchButton("OK");

//Performs a touch on the 1 item of the 'navigationbar0' control.
Aliases.Device.processPlrMeasureGo.window0.navigationbar0.TouchItem(1);
```
// Performs a touch on the 'buttonCrPlan' button.
Aliases.Device.processPlrMeasureGo.window0.buttonCrPlan.TouchButton();

// Contents of the edit control, as a text string.
Aliases.Device.processPlrMeasureGo.window0.scrollview0.textfield1.wText = "TestiOS2";

// Simulates a click on a control at the specified point (the coordinates are relative to the
// control). By default, a click is simulated in the center of the control.
Aliases.Device.processPlrMeasureGo.KBKeyView("Return-Key").Click();

// Contents of the edit control, as a text string.
Aliases.Device.processPlrMeasureGo.window0.scrollview0.textfield2.wText = "Leinfelden";

// Simulates a click on a control at the specified point (the coordinates are relative to the
// control). By default, a click is simulated in the center of the control.
Aliases.Device.processPlrMeasureGo.KBKeyView("Return-Key").Click();

// Performs a touch on the 'button' button.
Aliases.Device.processPlrMeasureGo.window0.scrollview0.button.TouchButton();

// Simulates a click on a control at the specified point (the coordinates are relative to the
// control). By default, a click is simulated in the center of the control.
Aliases.Device.processPlrMeasureGo.window0.actionsheet0.collectionview0
  .collectionViewcell0.actionview0.Label("Take a new photo").Click();

// Simulates a click on a control at the specified point (the coordinates are relative to the
// control). By default, a click is simulated in the center of the control.
Aliases.Device.processPlrMeasureGo.window0.actionsheet0.collectionview0.
collectionviewcell0.Click();

// Simulates a click on a control at the specified point (the coordinates are relative to the
// control). By default, a click is simulated in the center of the control.
Aliases.Device.processPlrMeasureGo.window2.ActionSheet().CollectionView() 
  .CollectionViewCell(0).Click();

// Delays the test execution for the specified time period.
Delay(2000);

// Sets the orientation of the device screen.
Aliases.Device.Desktop.SetOrientation(soPortrait);

// Simulates a touch on the view.
Appendix A. Appendix E

Aliases.Device.processPlrMeasureGo.window0.ImageView().Touch();

//Simulates a touch on the view.

Aliases.Device.processPlrMeasureGo.window0.butttonCrPlan.Touch(0, 0);

//Simulates a touch on the view.

Aliases.Device.processPlrMeasureGo.window0.Touch(156, 535);

//Performs a single touch on the specified button.

Aliases.Device.processPlrMeasureGo.window0.buttonUsePhoto.TouchButton();

//Performs a touch on the 'buttonCreate' button.

Aliases.Device.processPlrMeasureGo.window0.scrollview0.buttonCreate.TouchButton();

//Performs a touch on the 'button' button.

Aliases.Device.processPlrMeasureGo.window0.collectionview0.collectionviewcell0.button.TouchButton();

//Simulates a click on a control at the specified point (the coordinates are relative to the control). By default, a click is simulated in the center of the control.

Aliases.Device.processPlrMeasureGo.window0.collectionview0.collectionviewcell0.Click();

//Performs a touch on the 'button2' button.

Aliases.Device.processPlrMeasureGo.window0.scrollview0.button2.TouchButton();

//Delays the test execution for the specified time period.

Delay(1000);

//Performs a touch on the 'buttonCrPlan' button.

Aliases.Device.processPlrMeasureGo.window0.butttonCrPlan.TouchButton();

//Simulates a click on a control at the specified point (the coordinates are relative to the control). By default, a click is simulated in the center of the control.

Aliases.Device.processPlrMeasureGo.window0.actionsheet0.collectionview0.collectionviewcellUseGraph.Click();

//Simulates a click on a control at the specified point (the coordinates are relative to the control). By default, a click is simulated in the center of the control.

Aliases.Device.processPlrMeasureGo.window2.ActionSheet().CollectionView().CollectionViewCell(2).Click();

//Delays the test execution for the specified time period.
Delay(2000);

//Simulates a click on a control at the specified point (the coordinates are relative to the
control). By default, a click is simulated in the center of the control.

Aliases.Device.processPlrMeasureGo.window0.buttonLine.Click();

//Simulates dragging on the view.
Aliases.Device.processPlrMeasureGo.window0.Drag(173, 177, 338, 191);

//Simulates dragging on the view.
Aliases.Device.processPlrMeasureGo.window0.Drag(79, 134, 144, 112);

//Performs a touch on the 'buttonMsr' button.
Aliases.Device.processPlrMeasureGo.window0.buttonMsr.TouchButton();

//Enters text in the 'textfield0' edit control.
Aliases.Device.processPlrMeasureGo.window0.textfield0.wText = "12";

//Performs a touch on the 'button12' button.
Aliases.Device.processPlrMeasureGo.window0.button12.TouchButton();

//Performs a touch on the 'button6' button.
Aliases.Device.processPlrMeasureGo.window0.button6.TouchButton();

//Enters text in the 'textfield0' edit control.
Aliases.Device.processPlrMeasureGo.window0.textfield0.wText = "T";

//Simulates a touch on the view.
Aliases.Device.processPlrMeasureGo.window0.Touch(280, 428);

//Simulates a touch on the view.
Aliases.Device.processPlrMeasureGo.window0.Touch(100, 10);

//Enters the text 'This is working' in the 'textfield0' edit control.
Aliases.Device.processPlrMeasureGo.window0.textfield0.SetText("This is working");

//Performs a touch on the 'button7' button.
Aliases.Device.processPlrMeasureGo.window0.button7.TouchButton();
Appendix A. Appendix E

//Simulates a touch on the view.
Aliases.Device.processPlrMeasureGo.window0.Touch(454, 435);

//Simulates a touch on the view.
Aliases.Device.processPlrMeasureGo.window0.Touch(195, 286);

//Simulates keypresses on the view.
Aliases.Device.processPlrMeasureGo.window0.textView0.Keys("This is a sticky note");

//Performs a touch on the 'buttonDone' button.
Aliases.Device.processPlrMeasureGo.window0.buttonDone.TouchButton();

//Performs a touch on the 'button8' button.
Aliases.Device.processPlrMeasureGo.window0.button8.TouchButton();

//Performs a touch on the 'button8' button.
Aliases.Device.processPlrMeasureGo.window0.button8.TouchButton();

//Performs a touch on the 'button9' button.
Aliases.Device.processPlrMeasureGo.window0.button9.TouchButton();

//Performs a touch on the 'Back' item of the 'navigationBar0' control.
Aliases.Device.processPlrMeasureGo.window0.navigationBar0.TouchItem("Back");

//Performs a touch on the 'Back' item of the 'navigationBar0' control.
Aliases.Device.processPlrMeasureGo.window0.navigationBar0.TouchItem("Back");

//Performs a touch on the 2 item of the 'navigationBar0' control.
Aliases.Device.processPlrMeasureGo.window0.navigationBar0.TouchItem(2);

//Performs a touch on the 'Back' item of the 'navigationBar0' control.
Aliases.Device.processPlrMeasureGo.window0.navigationBar0.TouchItem("Back");

//Performs a touch on the 1 item of the 'navigationBar0' control.
Aliases.Device.processPlrMeasureGo.window0.navigationBar0.TouchItem(1);

//Performs a long touch on the 'button' button.
Aliases.Device.processPlrMeasureGo.window0.collectionView0.collectionViewcell0.button.LongTouchButton();
// Performs a long touch on the 'button2' button.
Aliases.Device.processPlrMeasureGo.window0.collectionView0.collectionViewCell1.button2.LongTouchButton();

// Simulates a touch on the view.
Aliases.Device.processPlrMeasureGo.window0.collectionView0.collectionViewCell1.buttonDlt.Touch();

// Simulates a click on a control at the specified point (the coordinates are relative to the control). By default, a click is simulated in the center of the control.
Aliases.Device.processPlrMeasureGo.window2.alertview0.collectionView0.collectionViewCell(1).ActionView().Label("Delete").Click();
Bibliography


[Coghlan, D, Brannick T. (2005)]: Doing Action Research in your organisation. IRL.


[Hughes Systique Corporation, n. a., 2013.] Test Automation Tools for Mobile Applications: A
brief survey.

[Myers, G. J. (1980)], Software Reliability, Mir, Moscow.


[Iilchenko, A., 2011]. The basic definition of automated software testing. [Online] Available at: http://blog.qatestlab.com/2011/05/05/the-basic-definitions-of-automatedsoftware-testing/


[Taipale, O. (2007)], Observations on software Testing Practice; Doctor of science thesis; Lappeenranta University of Technology.


[Dr. Winston W. Royce]: http://www.oxagile.com/company/blog/the-waterfall-model/


106
Bibliography


[Noman Ali]: http://www.socialhunt.net/blog/types-of-mobile-app/


[Kulyamin, V. (2008)], Software Verification Methods, Moscow, pp. 4-8, 29-35

[Robert Half] https://www.roberthalf.com/technology/blog/6-basic-sdlc-methodologies-the-pros-and-cons