Anoja Peethambaran

Automated Functional Testing Using Keyword-driven Framework

Helsinki Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Thesis

13 May 2015

Author(s) Title Number of Pages Date	Anoja Peethambaran Automated Functional Testing Using Keyword-driven Framework 86 pages + 2 appendices 13 May 2015
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Mobile Programming
Instructor(s)	Teppo Jylänki, Project Manager Erja Nikunen, Principal Lecturer

The purpose of this study was to improve the software quality efforts by using software test automation. This proof-of study was carried out for Natural Resources Institute Finland (Luke Oy.) which serves as a research institute operating in three sectors namely forest, agriculture and food industries, and game and fisheries. Test automation is performed mainly for mobile web apps.

An action research methodology was used in this study. A qualitative study was conducted to identify an appropriate tool for automation by analysing the features using 11 questions published by TestLab4Apps. Quantitative study was used to analyse the results of the research to verify the improvements having taken place.

The results show that there is a tremendous improvement in the quality of the software delivered after using test automation. In addition, it speeds up the release cycle times and delivery can be made in a short span of time without affecting the quality of the software. Also the test scripts created could be maintained, in future, by non-technical staff and thus make return-of-investment for the company.

The study recommends choosing test automation for other test cases also, as it benefits the company in the long run. The study was made for just mobile automation, but while conducting the study all the choices related to choosing the right tool were made by having also web application testing in mind. In this regard, for full return-of-investment the company can automate their web application in the same way, too.

Keywords	Mobile test automation, CI, Jenkins, Robot framework, Appium

Contents

Appendices

List of Figures/Tables

Table of Figures

List of Tables

Abbreviations/Acronyms

ADO	ActiveX Data Objects
API	Application Programming Interface
AUT	Application-under-test
App	(Mobile-) Application
BDD	Behavior Driven Development
CI	Continuous Integration
CM	Configuration Management
CSS	Cascading Style Sheets
CSV	Comma-separated Values
DAO	Data access object
HTML	Hyper-Text Mark-up Language
IE	Internet Explorer
IDE	Integrated Development Environment
JSON	JavaScript© Object Notation
MBT	Model Based Testing
ODBC	Open Database Connectivity
OSS	Open-source software
PDF	Portable Document Format
QA	Quality-Assurance
ROI	Return on Investment
SDLC	Software development life cycle
SCM	Source Control Management
SuT	System under Test
TDD	Test-Driven Development
UI	User Interface
VCS	Version Control System
WAR	Web Application Archive
XLS	Microsoft Excel file format
XML	Extensible Mark-up Language
XPATH	XML Path Language

Glossary

Agile	Group of software development methods focusing on the two key criteria that sets them apart from the traditional view of software process (The Waterfall Process): adaptive planning and a people-centred approach.
Bugs	A Software Defect / Bug is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectations (which may not be specified but are reasonable).
Configuration Management	Server that is responsible for tracking and controlling code changes.
Defect Tracking	System which one uses to track defects, feature requests, and any other tasks related to making changes on a system.
Software Development	The act of creating and maintaining applications involved in a software release life cycle and this will ultimately result in a software product.
Software Deployment	The process that makes a software system to be ready for use is termed as deployment. Sometimes it involves some activities to be performed at the manufacturer side, otherwise at customer side, sometimes from both parties.
Integration	A Software technique
Open-source software	Software that is available in market which is free to use
Production	Refers to the location to which the system is deployed, and which customers currently use.
Quality Assurance	It is a way of controlling the defects in software before it is delivered to the customers by means of monitoring the software engineering processes and methods used for its development.

Scrum	Iterative and incremental agile software development framework
Software	In simple words, it is computer instructions. System software consists of operating system and other utilities that will make the computer system to work. In other hand, application software is the one that perform the real end-user support like spreadsheet, word processors.
Staging	Server which is at a team accessible location and that is as close to production as possible

1 Introduction

Software has been part of modern society for more than half-a-century and in this era, many kinds of software with many features are available in the market competing each other. This ever-increasing competition forced the software to be released in a short time scale, making the software delivery time shrink from months to days or even to hours. For instance, in 2011 Jon Jenkins (Velocity Culture 2011), at that time a director at Amazon.com, announced that Amazon was deploying every 11.7 seconds. This implies that competition is very tough and customers' expectations are also high. This demand combined with the competition forces software to be made available to the market in short span of time. So it is very crucial to improve its quality and reliability by means of approaches such as software testing.

The company for which this study was performed is, at the time of writing this facing a tough phase of meeting the quality expectations by keeping the time constraints. Improvements or changes to the software emerge frequently. These constant changes and improvements of the company's application are released within a short time span.

The company currently does not follow any testing processes. However, testing is performed by the development team itself. They find it an inefficient way of quality assurance. To keep pace with today's short product release cycles; this company needs an automated solution that enables to build modular and reusable test assets. This current research initiative provides a proof-of-concept study on how to improve the current dilemma of the company.

1.1 Focus and Objective

This study focuses on automated continuous integrated testing of mobile applications using Keyword-driven framework in an Agile Software Project. The initial objective was to provide a sufficient literature review on different testing approaches and study the feasibility of incorporating automated testing for mobile applications to the company. The outcome is a proof-of-concept showing how the implementation fits to the company.

Objective: How the company can effectively implement their quality assurance effort for mobile web applications.

In order to achieve this objective, three research questions were identified to be answered in this research process:

1) How an efficient tool and a test framework could be chosen for the automation?
2) How the chosen tool and framework could be incorporated in the continuous integration?
3) How the chosen tool and framework could be implemented in end-to-end testing?

By analysing the implementation study some recommendations are proposed. The company can consider these recommendations while making a decision on employing automated testing in their release cycle. The scope is limited to the integration testing of company's web interface as well as mobile application.

1.2 Research Setting

The study was conducted for a company operating in three sectors namely forest, agriculture and food industries, and game and fisheries. The company serves the public through its research and also through the software applications running in web-browsers as well as in mobiles. One of the company's missions is to retain its customers in Finland and gain customers from other countries also. In order to fulfil this mission, the company aims for bug free software in minimal time. This mission fulfilment urges some improvements in the company's testing process and is the stage for this research.

The steps followed to conduct this research are depicted in Figure 1. Each one of the stages in the research steps is explained in detail below:

1. **Background study:** This phase is to collect all relevant information regarding software testing by reading publications and research papers. This form the basis for the current research work.
2. **Automation scope definition:** In this step it is important to know how the testing is implemented currently. By knowing the current implementation and testing scenarios, it is easy to prioritize the most important test cases for automation.

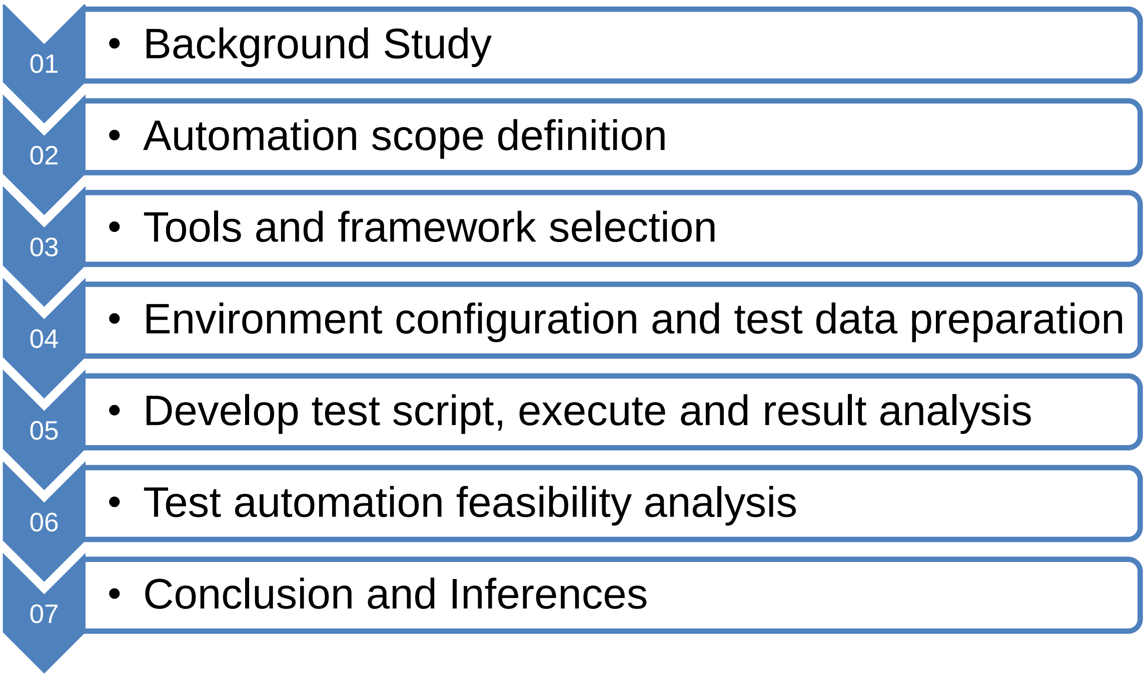

01
Background Study
02
Automation scope definition
03
Tools and framework selection
04
Environment configuration and test data preparation
05
Develop test script, execute and result analysis
06
Test automation feasibility analysis
07
Conclusion and Inferences

Figure 1: Research steps

3. **Tools and framework selection:** Once after selecting the important test cases and knowing the technology stack, the next important step is choosing the right tools and suitable framework for the automation.
4. **Environment configuration and test data preparation:** Next step after deciding on the tools and framework is setting up the environment for the automation task. This includes all the software installations and also the required test data preparations.
5. **Develop test script, execute and result analysis:** Upon setting up the environment, next activity is to develop and execute the test scripts. Being the crucial step, after execution the results are analyzed carefully for further comparison.
6. **Test automation feasibility analysis:** In this step an evaluation is performed for verifying if the tool supports automation of the application. The evaluation is performed by identifying how beneficial is automation comparing to its manual counter-part.
7. **Conclusion and Inferences:** Finally a conclusion is drawn regarding if the company can utilize the benefits of automation testing or how the company could proceed further with automation.

Each of the above stages were conducted sequentially so that the output of one stage forms the input of the next stage. Stage 1 reviewed all the theories related to testing and test automation. Stage 2 studied the present state of the problem at hand. Stages 3 and 4 prepared the environment for testing by choosing automation tool, framework and test environment. In Stage 5 the real implementation took place and Stage 6 analysed the results. From the analysis report, conclusion was driven in Stage 7.

2 Theoretical Background

This section details the literature associated with the automated mobile testing. As the published literature is analogous to the foundation of a building this section can be considered as the foundation for the entire research. A mind map as shown in Appendix 1 was created to identify the topics that can act as the foundation on which the research could be build.

2.1 Testing: Brief Overview

Testing is a process of executing a program with intent of finding an error (Myers 1979). Testing is the process of validating a system by executing the system and comparing the execution results with the actual requirements. This demonstration uncovers error and increases the confidence on the system under test. Thus it gives the assurance that the software will perform as it should be in its specified environment. Hence its goal is to improve the quality and reliability of the software.

Traditionally software testing was considered as an irrelevant effort for any software project. More specifically, during the early stages of software introduction, testing was viewed just as an activity to demonstrate that software is running and is working in right manner. In addition to this, testing was performed only if there were enough resources available and time. Otherwise, testing was treated as an unnecessary expenditure with respect to time, resource and money. Also cost of identifying and fixing bugs early reduces the cost comparing to if the bugs appear in the later stages of the production. Eventually the attitude towards testing changed after serious problems were reported due to lack of testing.

One such instance occurred during the summer of 1999 with the introduction of a new computerised passport processing system in two of the United Kingdom Passport Agency’s six offices - Liverpool and Newport. The new system failed to operate causing a huge backlog of applications waiting for processing. Hundreds of people had to drop their vacation plans and compensation was in the range of millions in addition to the overtime for staffs. (The UK Passport Agency report 1999)

Another accident resulted because of the inefficient testing is with the computerized radiation therapy machine called the Therac-25. Radiation overdoses caused by a fault in the control software of the device resulted around six deaths and several injuries to others. This could have been avoided by a thorough test. (Leveson & Turner 1993)

Such prominent incidents revealed the significance of testing. As computers and software are quite common in this technologically advanced world, it is quite natural that the face of testing has changed enormously. Currently software testing is an integral part of the software development process and it weighs high in any of the software projects. No more software is available to market without sufficient testing.

Moreover, some factors such as globalization increased the complexity of software products and the complexity is increasing day-by-day. The manufacturer has to spend money to ensure that the target audience will be satisfied with the software product. Also the complexity combined with the competitive pressures, demand high-quality software to be available in market in a short period of time. This puts forth the challenge of uncovering the defects as early as possible with minimum amount of time and effort. These demands put software testing to new heights as it can guarantee the software’s efficiency, reliability, integrity, portability, capability, usability, maintainability and compatibility.

Testing exposes the hidden bugs and well-tested software could ensure the quality of the product available to the market. Hence, testing is not just detecting the bugs, but it is regarded as a separate discipline to ensure the software quality. Some of the terms that are frequently used in software testing are error, fault, bug, failure and defect. They are represented in Figure 2 and their definitions are given below:

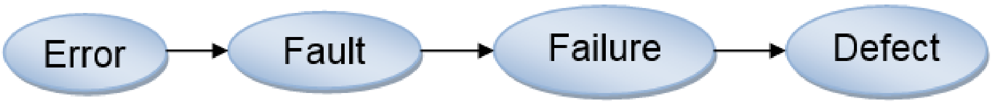


Figure 2: Common terms in testing

- *Error*: According to IEEE Standard 610.12-1990, an error is “a discrepancy between the computed, observed, or measured value or condition, and the true specified or theoretically correct value or condition”. A deviation from actual and expected value is considered as an error and it usually is a mistake made by humans.

- *Fault*: According to IEEE Standard 729-1983, a fault is “an accidental condition that causes a functional unit to fail to perform its required function”. It occurs because of the error. Faults are often called bugs.
- *Failure:* According to IEEE Standard 610.12-1990, a failure is “the inability of a system or a component to perform its required functions within specified performance requirements”. A failure is defined as a deviation of the software from its expected delivery or service, while the cause of such a failure is a fault (Grindal & Lindström 2002). Thus a fault causes a failure.
- *Defect*: According to Florac (1992), a defect is defined as “any flaw or imperfection in a software work product or software process”. Another definition by IEEE Standard 982.1-1988 is defects are “product anomalies such as omissions and imperfections found during early life-cycle phases and software faults”. Whenever failure occurs defect is also said to pop-up.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test (Kaner 2006). The definition and meaning of testing got lot of improvisations from early 60’s till date. The reason behind this is that testing was not considered that important during the early 60’s than it is now.

2.1.1 History

History of testing can be dated from 50’s, when FORTRAN, the first modern programming language was designed by John W. Backus. Gelperin & Hetzel (1988) classified the testing stages as below:

- Until 1956 – Debugging oriented period: During this period testing was often linked to debugging and there was no clear difference between testing and debugging.
- 1957 – 1978 – Demonstration oriented: During this period testing and debugging got clear distinction. Also the aim of the testing was to show that the software satisfies the actual requirements.
- 1979 – 1982 – Destruction oriented: On coming to this period the goal of testing was to find errors.

- 1983 – 1987 – Evaluation oriented: The aim of testing during this period is that during the software lifecycle a product evaluation is provided and measuring quality.
- 1988 – 2000 – Prevention oriented: During this period the tests were conducted to reveal that software satisfies its specification and also to detect faults and to prevent faults.

The above five are identified by Gelperin & Hetzel. However Extreme software testing (2009) identified one more stage i.e. automation oriented.

- 2000 – on-wards – Automation oriented: This period marks the origin of automation testing. The first keyword-driven automation was performed in 2000 and from there onwards testing got a new phase.

2.1.2 Types of Testing

Testing can be performed either manually or automatically. Both types have their own advantages and disadvantages. So, selecting one particular test type depends on the project's budget, requirements, suitability, expertise and timeline.

Manual Testing: As the name hints, manual testing is executing the test cases manually by a human without any aid from scripts or tools (Itkonen, Mäntylä & Lassenius 2009). That is, testing is performed by a tester acting in the role of an end-user. The completeness of testing is ensured by creating test plans, test cases and test scenarios. This type of testing is more suitable for usability testing, exploratory testing, ad-hoc testing etc.

Automation Testing: The test cases are executed with the help of an automation tool, script and software. That is, the tester first identifies the test cases. Automating test execution requires a form of test scripts that can run without human intervention (Henry 2008). Thus after identifying test cases test scripts are created for the test cases. Then these test scripts are executed using automation software. Once the test script is ready the test can be executed several times without taking much time and resources. Test automation is "no silver bullet either but it has a lot of potential and when done well it can significantly help test engineers to get their work done" (Fewster & Graham 1999). This testing type is mostly used for regression testing, performance testing, load testing and also for the test that needs repeated execution. Table 1 lists benefits of automation testing. The list is by no means exhaustive.

Table 1: Benefits of Automation (Software Testing Mentor 2015)

Manual Testing	Automation Testing
Tedious and time consuming: It is too tiring to test manually and will consume a lot of time because the tests are carried out by a human.	*Fast*: As this test is carried out by software, one can run it several times and the execution will be fast as well.
Human resource skill: Manual execution requires more testers to do the job.	*Less human resource skill*: Automation tool executes the test and so less human intervention.
Reliability is less: Human errors can badly affect the exactness of the test.	*Reliability is more*: Software runs it in the same way for every runs.
No programming help is available: Sophisticated test could not be carried out.	*Programmable*: Sophisticated tests can uncover hidden information

The advantages of automation testing are more than manual testing. Table 1 lists some of the strengths. However, not all test cases can be automated. In this case, it is beneficial to adopt both manual as well as automation testing.

2.2 Testing Methods and Approaches

It is a known fact that it is not possible to find out all the bugs in a program. Thus testers need to follow some strategy to expose the maximum number of bugs. The various test methods and approaches reveal several kinds of bugs. By following these test methods and approaches, the tester can make sure that most of the bugs are uncovered. In this section the main testing methods and approaches are briefly described.

2.2.1 Testing Methods

Static vs. Dynamic Testing: In static testing the program is not executed as such. This method examines the real program code manually or by using some software testing tools. It is also regarded as the verification. This type of testing is performed using code review, walkthroughs, document reviews, inspection, and feasibility analysis. Usually developers perform this testing before any other type of testing to find out any code breaks, syntactical errors, and undeclared variables etc. (Kaner, Falk & Hguyen 1999) On the other hand, dynamic testing executes the programmed code with some selected test cases. Thus this type of testing interacts with the real system by giving an input, collecting its output and comparing it with the expected result. Thus it detects the defects by interacting directly with the system. A separate team is allocated for this type of testing and they do not need to know anything about the implementation part of the system under test (Kaner, Falk & Hguyen 1999). Executing the system from a debugger environment is one of the typical cases of dynamic testing.

The box approach: Another kind of testing method classification is based on box approach. Under this approach three testing methods are identified:

- *White box testing*: Other names for this testing are transparent box testing, open box testing, clear box testing, structural testing and glass box testing. Tester should be fully aware of the system implementation to perform this type of testing. Source code is also available. This testing method tests various paths, data structures, loops, system states and decision points (Sommerville 2001). This type of testing can uncover defects quickly comparing to black-box testing and the test coverage is also treated as complete. Compared to black box testing, this testing requires vast testing knowledge and also knowledge on some tools like source code analysers and debuggers.

  Advantages: Maximum test coverage and effective testing possible with source code knowledge.

  Disadvantages: High investment for skilled testers and specialized tools, as test cases increase some defects may go untested.
- *Black box testing*: As the name shows, the software is treated as a “black box”. In this testing method the tester is not aware of the systems internal structure or architecture or workings. Even source code is also not available. Thus the tester has only the knowledge of what the software is intended to do and is unaware of how the software does an activity. Thus the tester is unaware of the system implementation, but know only the system functionality and so is otherwise called functional testing (Sommerville 2001). This lack of knowledge of the internal functionalities makes the testing effort to take longer time. One typical example of this kind of testing is that an invader uses the applications.

Advantages: Fit in large code fragments, source code not needed, clear separation between users' and developers' outlook, testers' skills need not be excellent.

Disadvantages: Test coverage is limited as test scenarios is limited, testers do not have knowledge about the system and so this testing is not that efficient, it is hard to design the test cases.

- *Gray box testing*: A concept between white-box and black-box testing is gray-box testing (Kaner, Bach & Pettichord 2001). This type of testing is a combination of both white box testing and black box testing. Testers at least have some knowledge of the internal workings of the system and they must have gained the knowledge by reviewing architecture diagrams and detailed design documents. Test execution is performed at black-box level. Also source code is not fully accessible. One example for this type of testing is checking the database tables by querying after executing some test.

  Advantages: Benefits of both white-box and black-box testing, with less knowledge excellent test cases, instead of source code testers gain knowledge from system documentation, testing performed from user's perspective than a designer's view point.

  Disadvantages: Test coverage is limited as no full access to the source code, some program paths may not be covered.

Table 2 presents a comparison between black-box testing, gray-box testing and white-box testing.

Table 2: Comparison of black-box, gray-box and white-box testing

Black-box testing	**Gray-box testing**	**White-box testing**
No knowledge of internal workings	Limited knowledge of internal workings	Full knowledge of internal workings
End-users, testers and developers	End-users, testers and developers	Usually by testers and developers
Blind testing	Testing based on system documentation	Tester has complete system knowledge
Least time-consuming	Moderate time-consuming	Most time-consuming
Algorithm testing not supported	Algorithm testing not supported	Algorithm testing is supported

Trial-and-error test execution	Boundary testing and data domain testing possible	Boundary testing and data domain testing possible
Also called functional testing	Also called translucent testing	Also called code-based testing

Table 2 compares the three box approaches: black-box, grey-box and white-box testing. Among the three selecting a testing method depends on number of factors like time for testing, application resource accessibility etc. Depending on the system requirements select a suitable method.

Visual Testing: Visual testing is the process of validating the visual aspects of an application's User Interface (Carmi 2014). This kind of testing records the entire test process so that the recorded footages could be handover to developers in case of any failure. This recording in video format captures the screen pictures of user actions and audio commentary is also provided. This kind of evidence provides more clarity and understanding to the defect and developer can focus more to identify the cause of the fault. The quality of communication is better in this case than just describing the fault and thus this kind of testing is best suitable in an agile environment.

2.2.2 Testing Approaches

Test approaches are the guidelines to be followed for better results. They are also known as test frameworks. The guidelines can be about test-data handling, coding standards, object repository handling etc. On following the suggested guidelines the benefits can be higher portability, increase code re-usage, cheap script maintenance charge etc. One important thing to note is that they are not mandatory rules but just guidelines and if followed will bring better results. Various approaches followed are given in Figure 3:

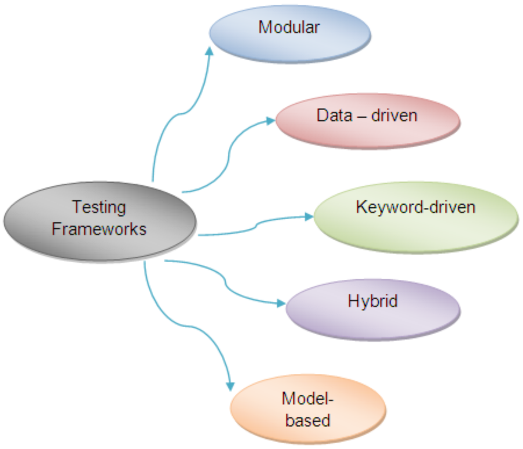


Figure 3: Testing Frameworks Types

Figure 3 above depicts the types of testing frameworks available. Each one has its own benefits as well as pitfalls. Next section explains each of the frameworks in detail.

Modularity driven testing: Out of all the other frameworks, this one is the simplest one to understand and to get proficient. As the name suggests, modules or functions or sections of the application-under-test (AUT) act as a representation for the test script creation. The created independent test scripts are just for a small portion or module of the system. These small test scripts are then used to make larger test scripts by using those in a hierarchical fashion and ultimately form one test case. (Kelly 2003) Figure 4 provides an example of modularity driven testing.

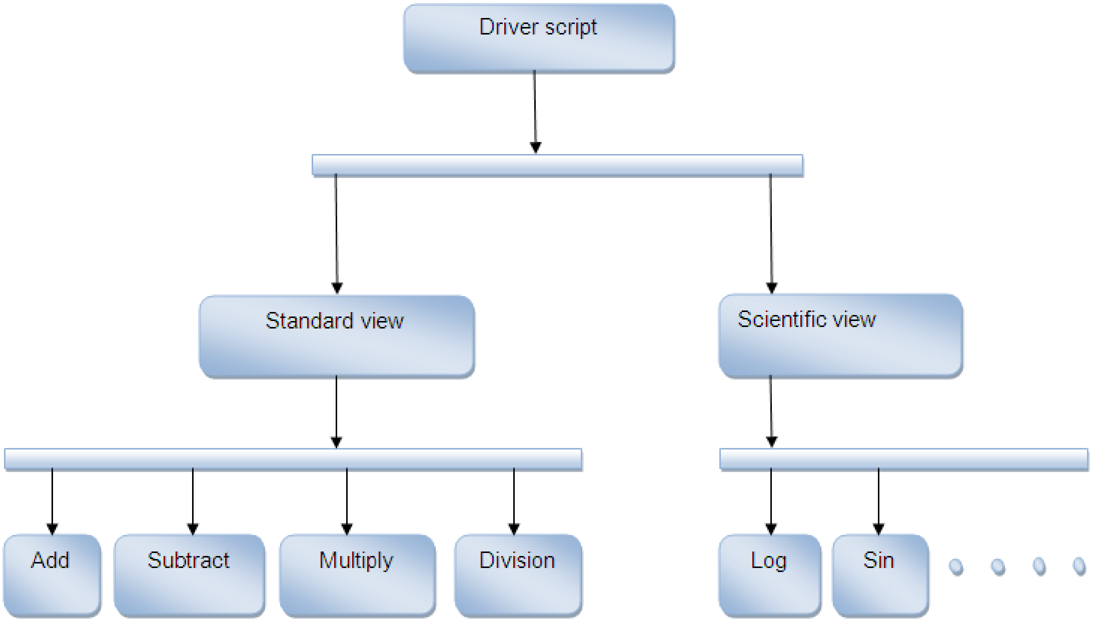


Figure 4: Example for modularity driven testing

Figure 4 shows an example of how modular frameworks operate. It shows scripts for the calculator program. Each rounded rectangle represents the scripts with driver script as the top level of the hierarchy. This driver scripts contains standard and scientific view scripts which in turn has small scripts for each module like add, multiply, subtract, log etc. This kind of modularity brings abstraction to each layer and so any changes in the functionality will be affected just a module and all other parts are free from any modifications.

Benefits: Application design and test script modularity, scalability and maintainability of automated test suites, new driver script creation for various test cases are super easy and fast because the functionality is accessible in test libraries.

Drawbacks: Test data is embedded in those small test scripts and so any changes to the test data needs some update in the script. It adversely affect larger test scripts and to overcome this pit-fall data-driven testing frameworks originated.

Data-driven testing: In this framework test data is separated from test scripts and test data is put (usually in tabular format) in some external files like text files, spreadsheets, csv files, DAO objects, ADO objects, ODBC sources etc. The test scripts will just contain the test case logic and for test data it will fetch from external sources into variables. In the same way as input values and verification values act through variables, input data and expected output resides in external files. This separation of data from logic makes it possible to run all test cases with multiple data sets using a single driver script. (Kelly 2003) The basic structure of data-driven testing is shown in Figure 5 below.

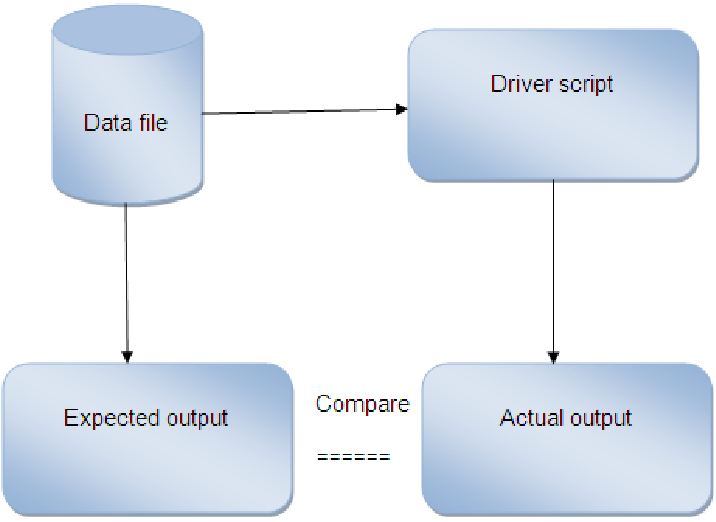


Figure 5: Data-driven testing

Figure 5 shows how the test scripts and external files connected to work together to form data-driven testing. Here the driver script contains all the test scripts and the navigation logic. It also includes the logic to read the data files and also logs execution status. This execution output could be then compared with the expected output that is stored in the external file. It is the duty of the driver script to read data file, script execution, output value computation and comparison with the expected outcome from the data file. As an example, the test script for the calculator program has two steps. The first step would be to prepare the test data file as shown in Table 3:

Table 3: Test-Data file for calculator program

Test Case	Number1	Operator	Number2	Expected Result
Add	5	+	2	7
Subtract	5	-	2	3
Multiply	5	*	2	10
Divide	5	/	-5	-1

Table 3 lists the inputs for the driver script and when the test scripts are executed the script loads these input data from the external file. The external file can be a database or excel files. However, the data is usually represented in a tabular format as given above. After preparing the test data next step is to prepare the driver script that includes the logic to connect this external file.

Advantages: Minimal test scripts needed as lot of the test scenarios could be covered by just modifying the test data, required minimal code only, bug fixing and maintenance is easy as test data and logic is separate, test data can be created even before UI is ready.

Disadvantages: New test cases require new driver scripts with new test data. Thus a change in either driver script or in data file requires corresponding changes in other one also. To overcome this hardship, keyword-driven testing frameworks popped-up.

Keyword-driven testing: This is also called table-driven testing framework and is an extension to the above mentioned data-driven testing framework. In this framework the test data and the keywords are kept in external files. The keywords are nothing but the action to be performed on the system under test like VerifyValue, InputText, and VerifyProperty etc. Though one needs to create data tables and keywords they are totally independent of the used automation test tool. This test tool helps in the execution of the keywords. Application is not required for test designing. The system functionality is written in tabular format and also in step-by-step instructions for each test. (Faught 2004) Figure 6 illustrates keyword-driven testing.

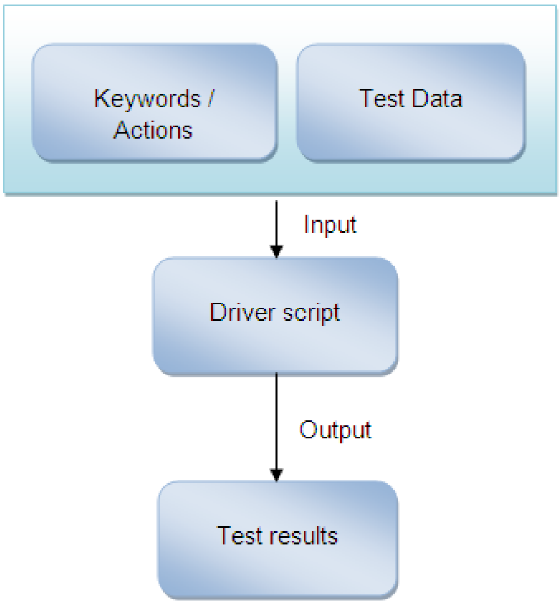


Figure 6: Keyword-driven testing

In Figure 6 it can be seen how keywords and test data flow to driver script. Driver script takes those as the inputs and does the script execution and then the results are logged. In addition to keywords there are two more components related to this framework. The first is Object Repository or Application Map that stores all of the objects that will be used in the scripts in a certain location instead of scattering them in the script. The second is Component Functions which are functions that work with GUI components. As an example the calculator program is tested using this approach. Again, one would first create the Data table as shown in Table 4:

Table 4: Data table for calculator program

Window	**Control (Object)**	**Action (Keywords)**	**Arguments**
Calculator	Menu		View, Standard
Calculator	Pushbutton	Click	1

Calculator	Pushbutton	Click	+
Calculator	Pushbutton	Click	2
Calculator	Pushbutton	Click	=
Calculator		Verify Result	3
Calculator		Clear	
Calculator	Pushbutton	Click	7
Calculator	Pushbutton	Click	-
Calculator	Pushbutton	Click	2
Calculator	Pushbutton	Click	=
Calculator		Verify Result	5
Calculator		Clear	

Each row in the data table represents each test step. The first column represents the application under test. The second column represents the control that shows the mouse clicks. The action column shows the mouse's action. Argument column shows the input value. After creating the data table, test scripts are created to read the data table and to execute each row one-by-one based on the keywords and then any error is checked and logs the result.

Benefits: In addition to all advantages of data-driven testing it does not require any automation expertise for test case creation and also the keywords could be re-used.

Drawbacks: Comparing to data-driven framework this is a bit more complicated and also requires complex test cases. In order to put together the strengths of all frameworks and thereby overcome the pitfalls, hybrid testing framework is introduced.

Hybrid testing: As the name indicates it is a combination of all the above testing frameworks: modular, data-driven and key-word driven, in order to achieve best results and to eliminate all of the downside. (Wright 2010) Figure 7 shows the steps taken in hybrid testing.

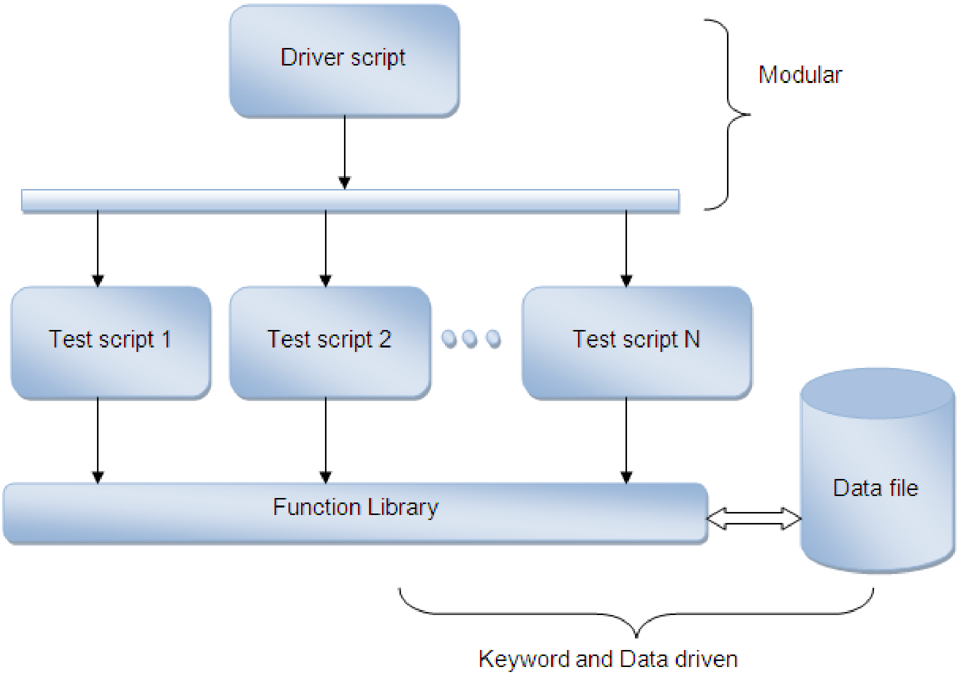


Figure 7: Hybrid testing

Figure 7 shows how to combine all of the testing frameworks. Thus data-driven testing can take the advantage of keyword libraries and the test scripts could be formed for modules for better abstraction. As an example one can take the calculator program again. Figure 8 illustrates the hybrid framework for the calculator program.

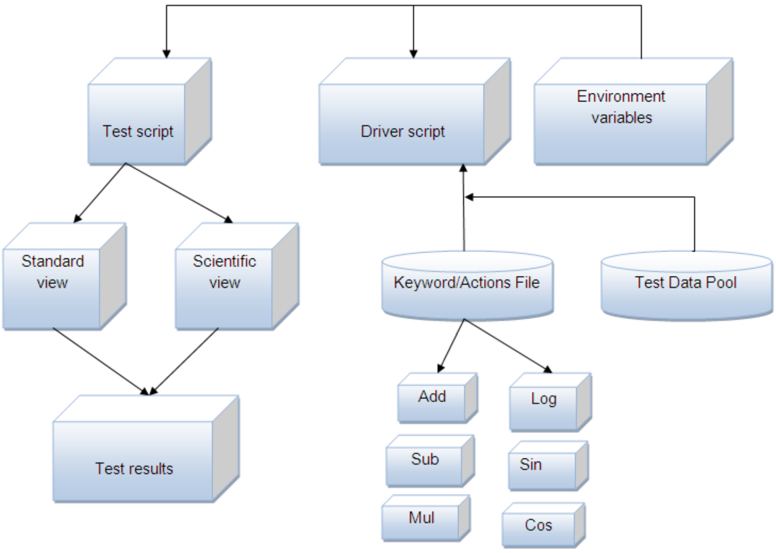


Figure 8: Hybrid framework for calculator program

As seen in Figure 8, it is evident how to merge modular, data-driven and keyword-driven frameworks. By doing in this way, one could eliminate negative sides of most of the frameworks.

Model based testing (MBT): In this software test procedures are automatically generated from the models of system requirements and behaviour. This testing approach produces test cases partially or fully from a (computer-readable) model that portrays certain features of the system under test. Further it should be carried out to enable test generation automatically or semi-automatically. If system's desired behaviour is modelled with some level of abstraction then it is called system model driven testing. On the other hand, if testing strategies are used for modelling then it is called tester model driven testing. (Utting & Legeard 2006) Figure 9 shows these testing approaches:

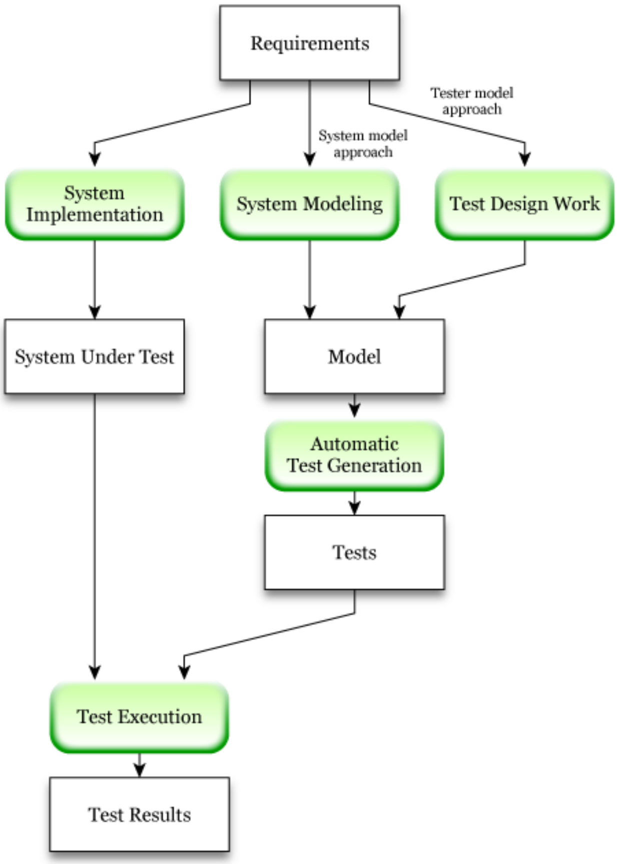


Figure 9: Model based testing (Conformiq 2014)

In Figure 9, system model driven testing and tester model driven testing is noted correctly. System models are like functional requirements and are little difficult to implement. These models could be reused. However, tester models are generated from the tester's traditional thinking and so they could not be reused. But tester models are easier to implement and are cheaper, though their benefit is little lower than system models.

Advantages: Minimal effort as test design is performed automatically, as a machine can uncover complex test scenarios than a human superior quality of test can be expected, test suite maintenance is trouble-free. Table 5 shows a comparison on the advantages and disadvantages of the different frameworks described above.

Table 5: Framework comparison

Approach	Advantages	Disadvantages
Modular testing framework	Module based abstraction Hierarchical structure Functions could be reused	Software dependent test script and test data in script makes it hard to reuse
Data-driven testing framework	Easy to maintain Not that complex	Software dependent test script and script creation needs skilled resources
Keyword-driven testing framework	Not dependent on software, maintainability and scalability	Script creation needs skilled resources and also needs much effort to create the scripts. Scripts can become complex also.
Hybrid testing framework	Combines strengths of all above approaches.	More complex as it combines all frameworks.
Model based testing framework	Testing with models, models could be reused, automatic generation of test	Complexity

Table 5 compares the available test frameworks. It shows that all of them are associated with strengths as well as weaknesses. So choosing the right framework depends on the project and the resources available.

2.3 Levels of Testing and Testing Artefact

Software development life cycle (SDLC) consist of various phases such as requirement collection and analysis, design, implementation, testing and deployment. Testing is performed in all of the stages of SDLC to avoid the fixing of error at the last stage. Testing level is identified by knowing at which phase of SDLC the testing is performed. Testing also requires some documents to be created as part of the process like test plans, test scripts, test matrices etc. They are termed as artifacts or artefacts. This section explains on the different levels associated with the testing and artefacts.

2.3.1 Testing Levels

Tests are basically classified based on at which phase of the software development life cycle the test is conducted. The grouping of these different levels is based on either by the test target or by the testing objective. Table 6 shows the testing levels.

Table 6: Testing levels

Test Level	Description
Based on test target	
Unit testing	It is performed on individual units or functions of code by developers to show that individual parts meet the requirements. This is performed before the code is moved to test environment. A unit test is also called a module test, because it tests the individual units that comprise an application. (Bentley 2004)
Integration testing	It is performed after integrating two components to show that the components work correctly as per the requirements even after integrating. Integration testing follows unit testing and precedes system testing. Integration testing is specifically aimed at exposing the problems that arise from the combinations of modules; so-called module interface errors. (Sommerville 1996) It can be done in two ways. In bottom-up approach testing is done first to low level components as in unit testing and then progresses the test to higher level components. In top-down approach the top integrated modules are first tested and then test all the modules down till it reaches to the low-level components.

System testing	It refers to testing the whole system after integrating all the components to check whether the system meets its requirements and the required quality and is done by allotted testing team. (Gittens, Lutfiyya, Bauer, Godwin, Kim & Gupta 2002)
System integration testing	It is performed after integrating the system to any third-party systems.
Based on test objective	
Installation testing	It tests whether the system is installed correctly at customer's hardware.
Compatibility testing	It tests if the application is compatible with all operating system versions, other software, target environment etc.
Smoke testing	It is performed after software build to make sure that critical functionalities like 'application starts successfully or not?' is tested to determine whether to reject a non-functional application to proceed to QA.
Sanity testing	It is performed after software build with minor changes like a bug fix to ensure that the fix is working and no other issues have popped-up.
Regression testing	It is performed after a major code change has happened to ensure that the change did not break any working sections. Main phases in regression testing are identifying feasible test cases and all collected test cases are then executed against the target (Holopainen 2004).
Acceptance testing	It is performed between two phases of the development cycle to ensure that application meets client requirements and is performed by QA team. The system is tested with real data rather than simulated test data. Acceptance testing may reveal errors and omissions in the original system requirements definition. (Miller & Collins 2001)
Alpha testing	It is the first stage of testing by developers or QA teams. Unit testing, integration-testing and system testing falls under alpha testing.
Beta testing	It is pre-release test after alpha testing and is done by selected group of target audience.

Destructive Testing	It tests with invalid or unexpected inputs making the system fail to ensure that the software works properly in those circumstances also. It basically tests for input validation and error-management routines.
Software Performance Testing	It tests for the performance of the system that can be caused by database transactions, client side processing, network delay etc. It can be divided as Load testing and Stress testing.
Usability Testing	It checks if the UI is easy to understand and use. Main objective is to find out any improvements.
Security Testing	It tests the system from security point of view to identify any flaws in the system regarding confidential data to avoid any intruders.
Functional vs non-functional testing	Functional testing tests a particular functionality. Non-functional testing tests non-functional attributes like performance, security etc.

Table 6 shows all the testing levels classified based on the test target and based on the test objectives. All of these fall under either functional or non-functional testing.

2.3.2 Testing Artefact

Artefacts are tangible by-product created during the SDLC. Testing artefacts are the documents that aid in testing process. Various testing artefacts are listed below:

- *Test plan*: This is a test specification that details the test strategy, test environment, any limitations, resources used for testing, schedule of testing, list of test cases and features etc. This information is very valuable for developers so that they can be careful while developing the system to pass the test cases. Test strategy is a higher-level document that will be used by some organizations.
- *Test case*: This document includes the set of steps, inputs and conditions to pass or fail a test. The means of deciding the pass or fail of a test is known as a test oracle. The term test oracle refers to a mechanism (e.g., a document or piece of software) that is used to determine if the result of an executed test is correct and test passed or not (Burnstein 2003).

- *Test Scenario*: This consists of several test cases with its sequence of execution to test a particular area or scenario.
- *Test script*: This represents the user actions as a program code. They are created using the test cases.
- *Test suite*: Multiple test cases together form test suite.
- *Traceability matrix*: It is also called Requirement Traceability Matrix (RTM). In this document each of the requirements is correlated to its corresponding test case. It can be used to trace the requirements for coding or to trace the test cases corresponding to a requirement for testing.
- *Test fixture or test data*: Data for testing a particular functionality is stored in a separate file called test data.
- *Test harness*: Test data input and output, test software, tools and configurations are together termed as test harness.

The above mentioned testing artefacts are created before or during the testing process. These artefacts are usually produced by testing personnel. These documents serve as a reference manual for future testing.

2.4 Testing Process

Every development or testing process of software follows an approach through which the developers or testers proceed their work and are called Software Development Process Models. To ensure success process models need to pass through a series of stages. Numerous testing processes are prevalent in software engineering. Two of the popular ones are the waterfall model and agile model.

- Waterfall development model: The waterfall model has a sequential flow with each of the software development phases are progressed in a sequence like one phase is finished first with its documentation and then next phase is started by inputting previous phase’s results (Royce 1970). In this model testing is performed only after finishing the implementation part, but before customer delivery. However, in most of the cases this testing period is viewed as extra days to be used to make up any delays in the development cycle. This badly affected the quality of the system due to lack of proper testing time and also bug fixes are too costly at this phase. Hence testing need to be performed in each of the development stages and is important to start early in the development cycle. Thus agile model got introduced.

- Agile or Extreme development model: This is a test-driven software development model. It has short and rapid test cycles starting early in the development life cycle. So bug detection, bug fixing, change requests etc. happen early and frequently. Also testing is performed throughout the life cycle process. Thus the testing quality is efficient as defects could be detected and fixed at any point of the development cycle making it not that expensive as well. (Schwaber & Beedle 2002)

From the above it is evident that in waterfall model it is not possible to go back to a completed stage whereas agile model allows this because agile is an iterative approach. It is very important to choose right model because the testing process greatly depends on the chosen model. However, agile model is the most preferred one these days.

2.5 Automated Testing

The testing process has become more and more complicated increasing the time taken for completing the testing levels. For instance, popularity of mobile devices is increasing day by day with several varieties of devices available to the market. These several devices differ in their size and also in their configurations. In this regard, testing performed manually may not be a good choice if changes happen frequently to the software product and time to market is very short. Here comes the benefit of automating the testing process. By automating one can speed up the time taken for testing and also can make more efficient tests.

Automated testing is one type of testing as mentioned in Section 2.1.2. It is done for applications of many types. This type of testing can be done for applications that run in a full Web browser on a desktop or laptop computer or testing can also be applied for applications run on mobile devices. Applications are of two types:

- *Traditional Web-application*: This kind of application is always accessed from a stable computer like a desktop one. During early days accessing an application in a mobile device was not that common as today. So applications created those days were intended for desktop based computers and such applications are generally termed as traditional web-applications. They are of two types: *Desktop-based applications and Web-based applications*. Desktop-based applications are applications installed on a desktop computer or on laptop, working locally. However, web-based applications are applications that could be accessed from a remote server using an internet in desktop computer or laptop. Such applications take the help of web browsers like IE, Chrome or Firefox.

- *Mobile based Application*: This kind of application is intended to run on small mobile devices such as tablets and smartphones, instead of a desktop or laptop computers. Such applications are designed by considering several constraints of the device such as its small size, various orientations, working platform etc. Mobile based applications are of three types:

  Mobile Native or On-Device Applications: Native apps which are installed through an application store are always present on the mobile device and could be accessed by tapping their icon. As they have their presence in the device, they could take advantage of the device features like camera, microphone, iPhone's accelerometer, audio etc. In addition to that each native app is intended for a specific platform like Android or iOS and so target users can install them on that particular platform.

  Mobile Web Applications: Web apps are mobile version of a web-site running on a mobile browser like Safari or Chrome. It looks exactly like a native app, but in reality it is not implemented and so it cannot be installed on the device. Thus they are not platform-specific and so could be accessed from any mobile platform. However, as it is not installed in the device these applications could not use the device's features.

  Mobile Hybrid Applications: A hybrid app is a combination of native and web apps. The native features allow it to enter into the application store and could be installed from there. Similar to the web app, they are also rendered in a browser using HTML; however the browser is embedded within the app. In other words, one can see the hybrid apps as wrappers for an existing web page; thereby minimizing the development effort and making a solid presence in the app store. It can be accessed by using the icons displayed on the mobile device and also it could use the device's features.

Mobile applications do not perform in the same way as their web-based counterparts because they are developed for various mobile handsets (Samsung, Nokia, HTC, Micromax) with different operating system flavours (e.g.: Android, iOS 4.x, iOS 5.x etc.) and with varying screen sizes and also with different hardware configuration like trackballs, hard keypad, virtual keypad etc. It is surely a challenging task to test such a system running in different handsets with differing OS and different screen sizes. So testing meant for web-based applications will not work for mobile-based applications. Henceforth, mobile apps require specialized testing strategies covering wide variety of devices with different screen sizes.

Robot framework: Many frameworks support mobile test automation. One such is Robot framework that is used in acceptance test-driven development (ATDD) and also in acceptance test. It follows keyword-driven testing with test data stored in tables. Though it has Python implementation, Jython (JVM) and IronPython (.NET) also supports it. It come up with a large collection of test libraries based on Python or Java and users can add new keywords. Nokia Networks stand behind its development and released under Apache License 2.0 as open-source with its source-code in GitHub. Its libraries and tools are also open-source and is operating system and application independent. (RobotFramework 2015)

Robot Framework provides a graphical user interface called RIDE (Robot Integrated Development Environment). Test case creation and management is so easy with RIDE. RIDE is considered as a complete IDE for Robot Framework with complete documentation.

Robot Framework’s integration to a Continuous Integration server is very simple. This is due to the script nature of the core Robot Framework system. Also for Jenkins, a CI server, a Robot Framework Plugin is available.

For testing it is essential to identify the Test-Libraries required for testing the AuT. For example, in order to test a web application Selenium2Library is used, but for testing iOS and Android apps. AppiumLibrary could be used. The list of Test-Libraries available is listed in Robot Framework’s website.

Some of the main benefits of using robot framework are that it has a very lively forum, extensive documentation is available, Keyword-driven testing, available keyword libraries are many, integration to CI servers possible. One negative thing can be installation requires many associated tools as well like RIDE, Python, Jython.

Continuous integration: Another tool that should be a help while implementing test automation is CI software. Continuous integration software helps to execute the tests automatically whenever the version control system identifies a change.

> “Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.” (Fowler 2006)

Whenever developers commit code changes to version control system, then the whole system is retested after integrating the updated system. Thus defects could be identified early and developers could do the fix immediately. This early identification of bugs makes it cheaper to fix them. Using CI stakeholders can make sure of the project’s status and progress. Figure 10 shows the CI – Workflow according to Bowes.

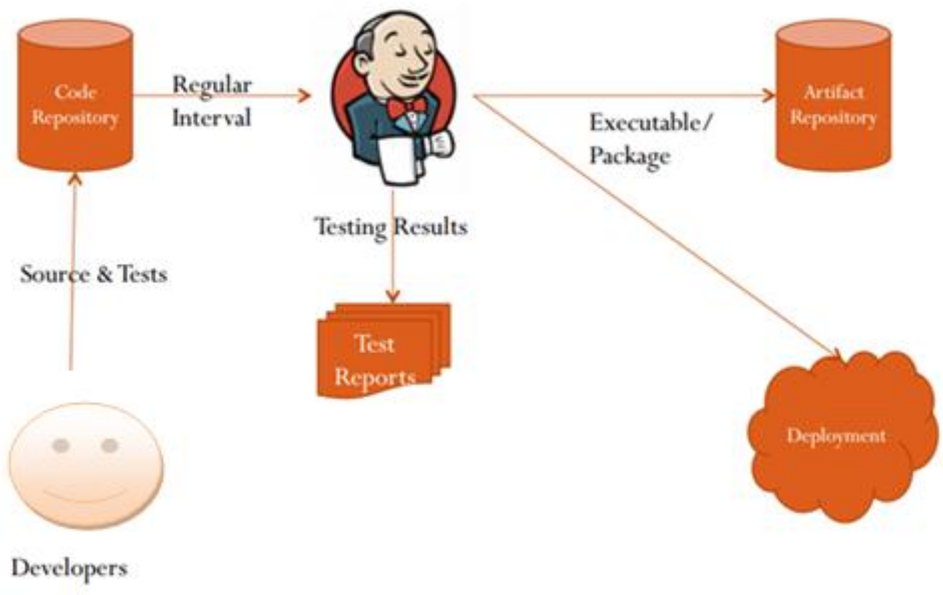


Figure 10: CI – Workflow (Bowes 2012)

Figure 10 depicts the normal workflow of a continuous integration cycle. This figure shows Jenkins as the continuous integration server. It is one of the most popular CI server widely available as open-source and is well supported by community. Developers make code updates in version control system. Jenkins regularly, in every minute or so, polls the version control system for any changes. Whenever a change is detected, the new version is checked-out to build-servers and new build is executed. After generating the executable file, all the test scripts are executed for bug detection. The results are reported back to Jenkins and notifications are sent to developers and stakeholders. Thus bugs, if any, could be fixed at the same time when bugs are introduced making Jenkins an efficient tool for ensuring software quality.

2.5.1 What/When/How to Automate

Full automation is impossible in most of the software. However, certain features could be easily automated like field validations, database connections, GUI components etc. Also automation should be applied only for certain circumstances such as:

- Complex, crucial and big projects
- Ample time for testing
- Minimal functional requirements change
- Stable software
- Frequent testing needed for certain areas

Another question while doing automation is how to perform automation. It is done with the help of a language like VB scripting and with the help of automated software. Some of the steps used to automate the testing process are:

1. Select the areas that fit for automation
2. Select the right tool for automation
3. Prioritize and select the test cases
4. Test script creation
5. Test suite development
6. Test script execution
7. Result report creation
8. Discover any defects or issues

These steps are followed sequentially for better results. These are some of the best practices followed in a successful testing and by following these steps brings maximum return on investment.

2.5.2 Automating Tools

Automation scripts can be created by using many tools such as the below:

- *Calabash*: functional testing tool for mobile apps
- *Appium*: allows to write functional **tests** to automate iOS and Android mobile apps
- *MonkeyTalk*: automates real, functional interactive **tests** for iOS and Android apps
- *eggPlant*: GUI test automation tool based on image recognition technology
- *TouchTest*: Mobile test automation for functional testing of native & hybrid apps

The above list is not an exhaustive list of available tools. The listed ones are just for illustration. The automating tools help to build and execute automated tests easily. Most of the tools provide record and playback feature allowing even non-technical personnel to perform the automated testing. Tools are designed to target certain test environment, such as Windows. Additionally they can be expensive depending on the provided features. Choosing a right tool is very important in testing process and is made by considering the requirements of the application under test.

2.6 Reflection on Literature Review

In this chapter, a detailed description of the basic concepts related to testing and also other tools that are useful for this research are reviewed based on the literature and available documentation. However, it is useful to show the aspects that are relevant to this research in a tabular format for better understanding and readability. Table 7 below serves that purpose:

Table 7: Literature summary

Section	Topic	Relevant Aspect
2.1	Testing: Brief Overview	
2.1.1	History	Automation oriented phase
2.1.2	Types of Testing	Automation Testing
2.2	Testing Methods and Approaches	
2.2.1	Testing Methods	White-box testing
2.2.2	Testing Approaches	Keyword-driven testing

2.3	Levels of Testing and Testing Artefact	
2.3.1	Testing Levels	System functional testing
2.3.2	Testing Artefact	Test case, test script
2.4	Testing Process	Agile development model
2.5	Automated Testing	
	Application Type	Mobile test automation for mobile web app and native app
	Test Framework	Robot framework
	Continuous Integration Server	Jenkins
	Automation Tool	Appium

Table 7 maps the theory reviewed with the concepts and tools used in this research study. Firstly it shows that the research falls under the automation oriented phase with the main topic of study as mobile test automation for mobile web app and native app. The testing conducted is system functional testing that falls under white-box testing and is using keyword-driven approach. The testing is conducted in an agile environment using Robot framework, Appium and Jenkins. Some of the documentation used for this testing is test cases and test scripts.

3 Technical Background

This section gives a brief introduction to the client company. This section also presents the company's Metinfo application to give the reader an idea about the software used for testing. In addition, a SWOT analysis is performed on the web application to find out the improvement areas. Finally, the current test process is analysed using metrics that could be used to compare the result after test automation.

3.1 Company Information

The research is conducted for Natural Resources Institute Finland (Luke Oy.). It is a research and expert organization. It promotes sustainable use of natural resources. It is formed in January 2015 by combining some of the state-owned institutes. The companies joined together are Metsäntutkimuslaitos, Riista- ja kalatalouden tutkimuslaitos

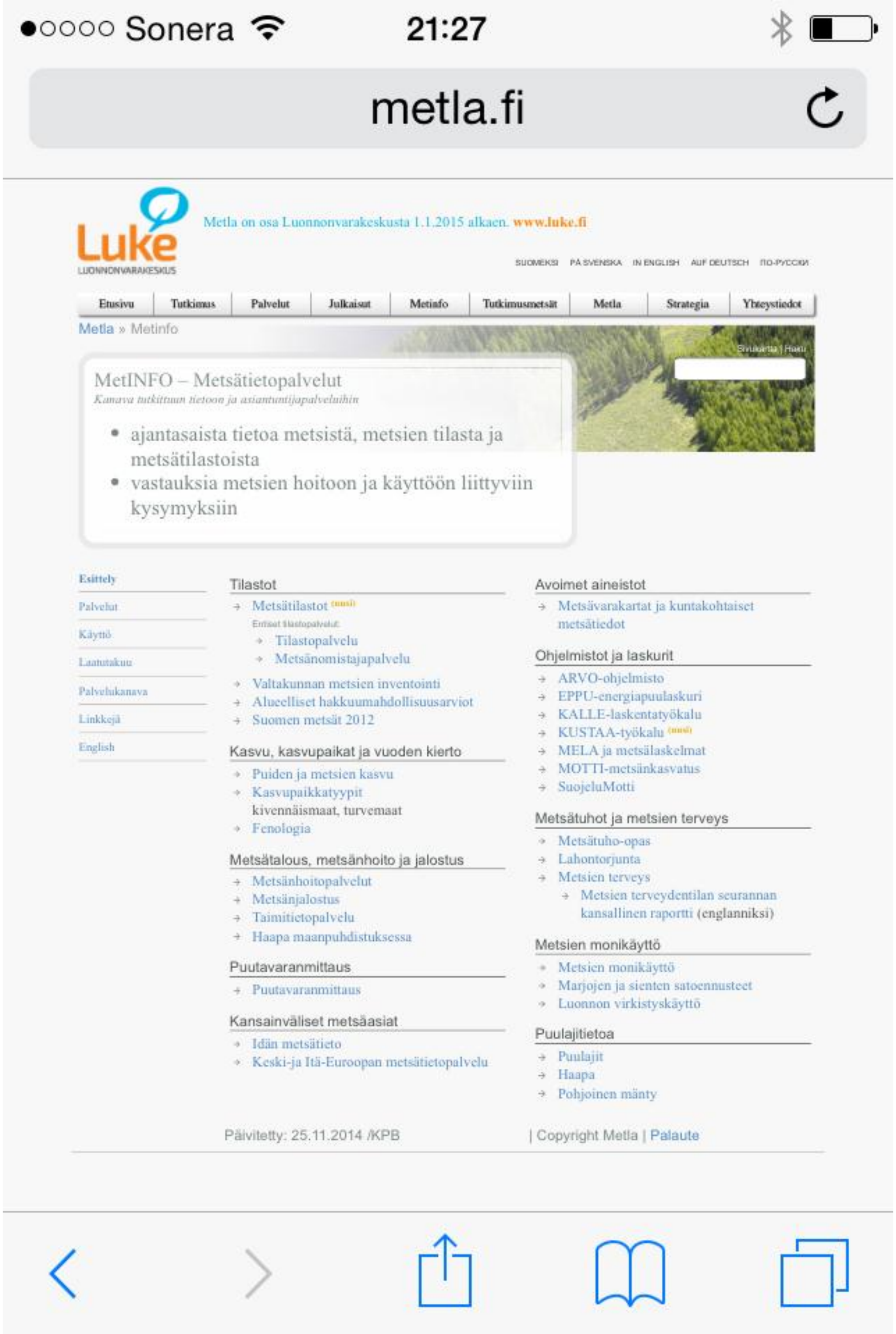
(RKTL), MTT Agrifood Research Finland and Tike. Thus its main sectors of operation are game, fisheries, forest, food and agriculture.

It supports social decision-making and produces statistics related to food and renewable natural resources. The research activities conducted help to build the bio-economy of the future. Its main service areas are natural resources information collection and processing as well as solution development for processing the information. Safe and correct data related to the natural resources are made available to the public via their web applications. There is possibility to retrieve previous year's data also for comparison purposes. Some of the software which could help to process the data is also available for the public to download and install. They also have some mobile native applications for feeding and storing data while they are not in company premises. For instance, while they are in forest feeding data related to the tree growth.

3.2 Application under Test

For the current research, Luke's Metinfo web application available in mobile was used. The data generated using the application is not considered for any of the analysis or assumptions. However, only the functionality of the applications is viewed for testing purpose.

Using the Metinfo application, users can directly use the collected data to generate some reports or they could use the data to analyze the current situation. Also users could directly download and install the available software on their device and can use it. Figure 11 shows the application used for the current study.



Metla on osa Luonnonvarakeskusta 1.1.2015 alkaen. www.luke.fi

MetINFO – Metsätietopalvelut
Kanava tutkittuun tietoon ja asiantuntijapalveluihin

- ajantasaista tietoa metsistä, metsien tilasta ja metsätilastoista
- vastauksia metsien hoitoon ja käyttöön liittyviin kysymyksiin

Tilastot
- Metsätilastot (uusi)
  Erilaiset tilastopalvelut:
  - Tilastopalvelu
  - Metsänomistajapalvelu
- Valtakunnan metsien inventointi
- Alueelliset hakkuumahdollisuusarviot
- Suomen metsät 2012

Kasvu, kasvupaikat ja vuoden kierto
- Puiden ja metsien kasvu
- Kasvupaikkatyypit
  kivennäismaat, turvemaat
- Fenologia

Metsätalous, metsänhoito ja jalostus
- Metsänhoitopalvelut
- Metsänjalostus
- Taimitietopalvelu
- Haapa maanpuhdistuksessa

Puutavaranmittaus
- Puutavaranmittaus

Kansainväliset metsäasiat
- Idän metsätieto
- Keski-ja Itä-Euroopan metsätietopalvelu

Avoimet aineistot
- Metsävarakartat ja kuntakohtaiset metsätiedot

Ohjelmistot ja laskurit
- ARVO-ohjelmisto
- EPPU-energiapuulaskuri
- KALLE-laskentatyökalu
- KUSTAA-työkalu (uusi)
- MELA ja metsälaskelmat
- MOTTI-metsänkasvatus
- SuojeluMotti

Metsätuhot ja metsien terveys
- Metsätuho-opas
- Lahontorjunta
- Metsien terveys
  - Metsien terveydentilan seurannan kansallinen raportti (englanniksi)

Metsien monikäyttö
- Metsien monikäyttö
- Marjojen ja sienten satoennusteet
- Luonnon virkistyskäyttö

Puulajitietoa
- Puulajit
- Haapa
- Pohjoinen mänty

Päivitetty: 25.11.2014 /KPB | Copyright Metla | Palaute

Figure 11: Luke's Metinfo Mobile Web Application

Figure 11 above shows the application used for this study. This application is mainly build for forest data collection and analysis. As seen in the screenshot the application as such is not a simple one. Thus testing involves several complex test cases.

The software development life cycle follows the agile development model with short release cycles. When a change request or bug report comes, the project manager prioritizes and enters the item into product log. During the next scrum meeting, with the approval from product owner, these prioritized requirements are selected for implementation and allocated for implementation.

After the implementation phase, the new version is deployed to the test environment. Potential end-users test the system and report any defects. After fixing the bugs from the test environment, the system is then deployed to acceptance testing environment. Here the testing is mostly to figure-out any configuration faults and regression test set will also be performed. After passing this testing, the system is moved to the production environment where the real end-users start using the system. Figure 12 illustrates the development process of application under test.

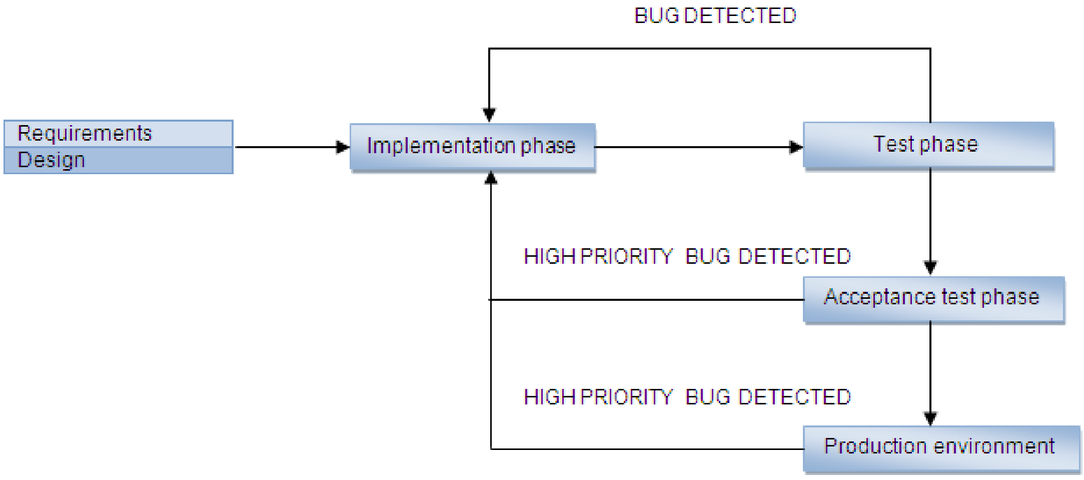


Figure 12: Development process of application under test

Figure 12 shows the development pipeline with each of the phases. It also shows the detected bugs' priority level at each of the testing phase. From the figure it is evident that bug fix is really important to pass to next phase in the process. Also only high priority bugs are identified in the acceptance test phase and also in the production environment. Another important point is that bug fixes to the later stages are very expensive.

3.3 SWOT Analysis

A SWOT analysis was conducted to analyze if the company’s software brings value to the business. By analyzing the strengths, weaknesses, opportunities and threats associated with the software product, the areas that need improvement could be figured out. It is a great tool that could be applied for any existing or new businesses to develop business strategies by considering the strong and weak points of the business, in addition to the opportunities and risk factors.

The name SWOT itself is short for Strengths, Weaknesses, Opportunities and Threats. The SWOT analysis is shown in Figure 13. It shows the acting force on the four quadrants. The top quadrants are organization’s internal attributes like location, patents, reputation etc. which could be improved by doing some effort. On the other hand, bottom quadrants are external attributes which resides in the market like technological change, sociocultural changes, competitors, prices, suppliers etc. which cannot be changed.

The SWOT analysis for the company was conducted by arranging a meeting with product owner and also project manager. First a brainstorming session was carried-out to identify elements in each of the four squares. Then for further development and clarification of the categories some questions were asked from each section. These questions are presented in Appendix 2. The answers to the questions were listed in the order of priority in each of the quadrants and are presented in Figure 13:

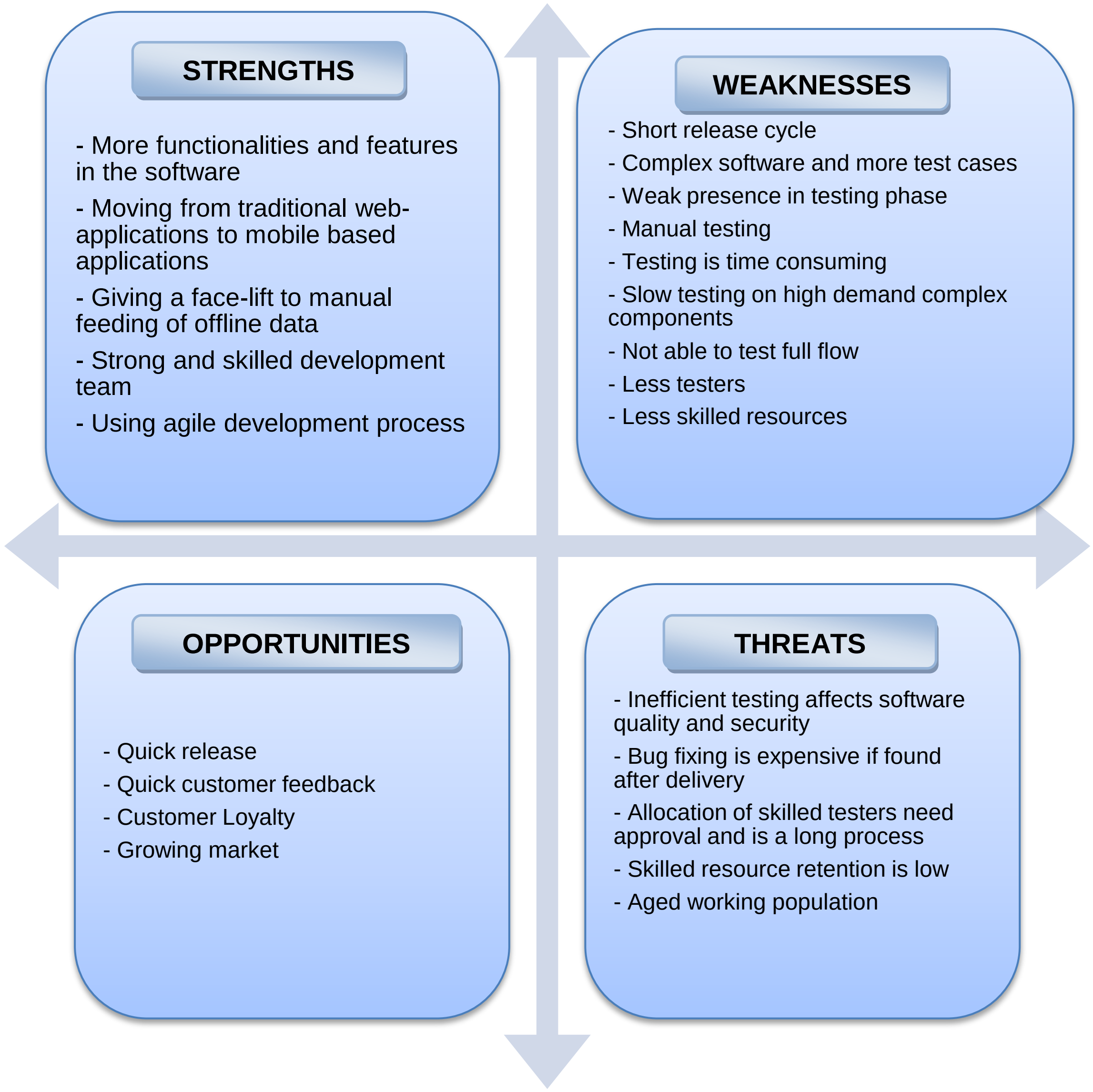

STRENGTHS
- More functionalities and features in the software
- Moving from traditional web-applications to mobile based applications
- Giving a face-lift to manual feeding of offline data
- Strong and skilled development team
- Using agile development process
WEAKNESSES
- Short release cycle
- Complex software and more test cases
- Weak presence in testing phase
- Manual testing
- Testing is time consuming
- Slow testing on high demand complex components
- Not able to test full flow
- Less testers
- Less skilled resources
OPPORTUNITIES
- Quick release
- Quick customer feedback
- Customer Loyalty
- Growing market
THREATS
- Inefficient testing affects software quality and security
- Bug fixing is expensive if found after delivery
- Allocation of skilled testers need approval and is a long process
- Skilled resource retention is low
- Aged working population

Figure 13: SWOT analysis of application under test

The above presented SWOT analysis clearly presents the business's favorable and unfavorable conditions that originate internally from the organization as well as externally from the target market. This helped to come up with a plan to effectively improve the business by incorporating some changes to the existing working conditions. The analysis is described in detail below:

- *Strengths*: The software is intended for calculations like how is the tree's growth, what is the rate with respect to harvesting etc. So it incorporates lot of functionalities and features to perform such complicated calculations. Traditionally, it was done manually and it consumed lot of time and also it contained lot of human errors. That is the reason why this application is introduced. Initially, the application was meant for traditional web applications and now they are moving to mobile based applications also.

The data collection from forest sites puts forth some limitation with respect to internet connection. This was the reason for the need of mobile based applications which work even without internet so that they could feed the data to the offline applications. When online researchers can transfer all these to a database and start their calculations and other processing. This face-lift from manual feeding to mobile-based collection brings forth added value to the application.

The development process followed is agile development model and so the software could be released in short cycles. The internal resources for development are very skilled in their respective fields. The technological stack of the application is very strong and builds with latest technologies.

- *Weaknesses*: As agile model is followed the release cycle is short and the time to test is also short. As the application grows the test cases also increases and among them some are complex. Testing the complex cases needs more time and tests are omitted if there is no modification on the time consuming complex test cases.

  Currently the testing is performed manually by developing team after the implementation phase. This also is another factor for the time consumption in the testing phase. Due to less skilled testers 100% testing is not possible right now and bugs pops up even after deployment. Going forward this is a great concern for the Metinfo application with respect to the sales and return of investment.
- *Opportunities*: Because of the agile process model company could make quick software releases with new features and bug fixes. This of course makes the customers happy and also makes them loyal.

  Currently the application is targeted to Finnish market and got good positive responses. Going forward the software will target foreign markets as well, retaining the existing Finnish customers.
- *Threats*: Customers are bit worried about the defects after delivery. Also bug fixes after delivery seems to be very expensive. The company is a state funded research organization. In this regard, any new resource funding will get approv-

al from top management and is a long process. It also takes long time to get everything in place.

In addition to that company faces tough time to retain the skilled resources. The reason behind this is as it is a state funded organization; the wages and potential skill level growth are low. Considering this fact if the resources find other opportunities they will go forward with that. Additionally, aged working population makes it hard to introduce new technology. Even transition of manual paper works to mobile platforms was not acceptable by many of the resources.

- *Potential strategies for growth*: From these above analysis, it is evident that the company needs improvement in their working environment. In order to deliver bug free quality software, company needs to put more effort on its testing phase. However, because of the funding issue and long approval process it is not possible to allocate efficient testers dedicated for testing. In this regard, test automation opens new door to the company. Though test automation could automate the testing process, it requires some resources to create and maintain the test cases. As the working population is aged and availability of skilled resources is less, it is important to do the test automation with right tools that could be used in future by the existing resources.

3.4 Analysis of Current Test Process

This section details the testing process in the company. The developers act in the role of testers after the implementation phase. Also they are not available full time to test the system because other development work will also go in parallel. For instance, a resource will be working on implementing some feature to the application and is also engaged in testing the application.

As there is no dedicated testing team, limited test planning is performed with respect to the test distribution as to what, when and how to do the testing. Also the existing test plans and test scripts seem to be outdated though test cases for some important test cases already exist. The main reason for not maintaining proper documentation is that resources are so much allocated with other tasks and they are not available full time for testing. This lack of time and less skilled testers makes it hard to plan the testing activities effectively.

The project manager also coordinates the testing activities. He will initiate the system testing and make sure that at least the latest modifications are tested well with the limited time and resources. The project manager prioritizes the bugs and enters them to the product backlog for fixing.

Due to limited time testing is not performed on the complete system. Only the latest modifications or added functionality are tested. As there are complex test cases testing the entire system needs more time. However, this strategy of testing just the modifications does not seem to be right. For instance consider the case of sharing one object with two flows. Figure 14 illustrates the shared object concept.

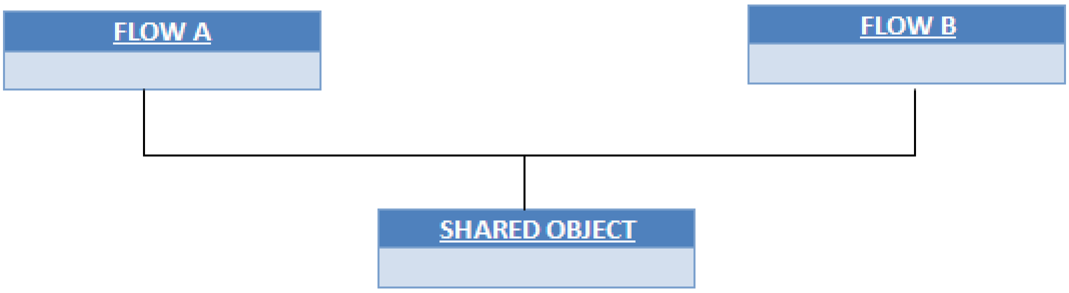


Figure 14: Shared object example

Figure 14 shows an object shared between flow A and B. Suppose flow A has some modifications that resulted in an unintentional modification to flow B also. Without testing the entire flow, those bugs will not be revealed till delivery.

Test automation provides the possibility to perform a full scale regression testing. Also automated tests created and maintained well, could be used further any number of times to test the system thoroughly and consistently even if design documents, test specifications, use cases and other test artefacts are not available. Thus test automation for the company will increase the system reliability by reducing these risk factors

4 Method and Material

Software development process by itself is divided into several phases such as requirement collection, designing, implementation, testing, shipping etc. Each of these phases involves complex situations to be dealt with where the participants are not sure of the nature of the problem and how to solve it. Also each software development project varies from one another and thereby solution for one project may not be applied to another, though their problem is same. In this regard, conduct a current state analysis of the problem and then plan and execute a change to solve the problem seems to be more valuable. This allows the stakeholders and researchers jointly plan, execute and solve an uncertain or complex problem. Henceforth, this thesis chooses action research as the main approach method used to conduct the study considering to the fact that this thesis subject is to identify solution to a particular problem for the company.

4.1 Action Research

The history of Action Research (AR) is considered as a complex one as it is not rooted from a particular field. Instead, its rise was occurred gradually over years from several fields. An American psychologist, Lewin (1946) founded action research following World War II. According to Lewin this research helps the researcher to gain knowledge related to a system while changing the system" (Elden & Chisholm 1993). Dick (2002) also has a similar definition for Action Research as action and research performed simultaneously in a spiral way.

In Action Research, the researchers attempt to work on an issue to get rid of the matter while simultaneously gain some understanding pertaining to the issue at hand (Davison, Martinsons & Kock 2004). The researcher totally turn as an essential element in the study (Jenkins 1985). On contrary to other empirical research methods that study a matter in its original form, action research study an issue by making changes to the current form (Easterbrook, Singer, Storey & Damian 2007). So Action Research triggers some change process instead of looking out for some generalizations or providing some theories to be correct or not. This kind of research is more suitable for the fields where some changes for a situational problem cannot be studied without implementing it. Software Engineering is one such field where action research can be applied effectively.

In this kind of research there must be a problem owner who ultimately turn out as a collaborator who is ready to participate in both studying and solving the issue. The researcher can also be the problem owner in certain cases. The quality of action research is identified by the reality and importance of the issue at hand and also by the authenticity of the outcome. (Easterbrook, Singer, Storey & Damian 2007)

Action Research is carried out by the researcher involving the problem owner also at every phase of the study thus making it a researcher oriented and participatory process (Bradbury-Huang 2010). To conduct this kind of research, the researcher must follow the Action Research Cycle, which is developed by Lewin (1946). His cycle is presented in Figure 15. This cycle consist of three phases: planning action, taking action and evaluating action. This cycle is iterated over many times in the research until the result is achieved.

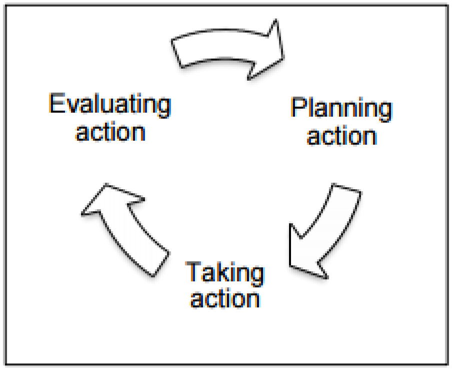


Figure 15: Action Research Cycle

In Figure 15 it is evident that each cycle of the research starts with a thorough study of the problem at hand. This rigorous study of the problem leads to furnish some plans to be carried out in order to solve the problem. This action plan must be executed as next step, and finally an evaluation has to be performed to check if the course of action has any effect on the problem.

Kemmis & McTaggart (2000) put forward a spiral model for action research that consists of four broad phases of iterating cycle which is depicted in Figure 16. Different stages in this model are plan, act, observe, reflect and re-plan. These stages are repeated in a spiral way till a satisfied outcome is obtained from the action.

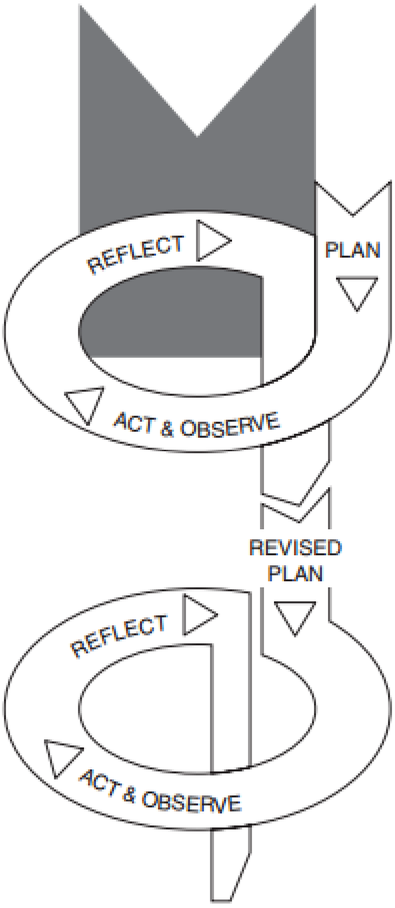


Figure 16: Kemmis & McTaggart’s action research spiral

In the model suggested by Kemmis & McTaggart the first step is Planning phase at which problem identification is performed. After the problem is specified a plan of action is developed in order to improve the situation. This plan is then executed carefully over a specified period of time by engaging in some interventions into the current work flow. Then the researcher observes systematically the effects of the action and documents it clearly. This observation phase can be considered as a data collection phase at which the researcher collects all information about what is happening. Finally evaluation of the happenings is performed by assessing the effects of the action. Based on this assessment result the researcher decided on whether to conduct further action research to improve the situation even more. This thesis uses this action research spiral of planning, acting, observing, reflecting and re-planning to improve the company’s situation.

4.2 Research Approach

First of all the author’s professional experience was mostly with software development. So understanding of software testing is quite vague. To be specific test automation is quite a new topic for the author. In this regard, action research could give more understanding on the topic as well as AR could fit more to the problem at hand.

It is also quite interesting to note that most of the research conducted in the field of software engineering uses action research as research methodology. The main reason behind this is that each research problem is unique and also solutions for such unique situations could be produced only by applying it to the real development environment. The main principle behind AR is to engage both researcher and stakeholders in the research work and so this thesis also interacts with the problem owners at each stage.

This research is performed in several stages. These stages are described below:

- First stage is to select a right tool and test framework for the test automation by comparing against some criterions.
- Second stage is to incorporate these selected tool and framework in the continuous integration
- Third stage is the real implementation part. In this stage the environment will be set-up then will identify the test cases and will do the execution.
- Fourth and final stage is to analyze the test result and put-forth the feasibility analysis of the test automation.

The list given above presents the different stages of the current research. All of these stages was quite important for the success of the study. Choosing right tool and framework was based on the system requirements. Continuous integration really automates the whole testing process till the test report generation and test feedback. After preparing the test environment and test data, real implementation is started and results were recorded for analysis. Finally the feasibility analysis was conducted based on the test results.

5 Tool Selection

In the market, choices available for test tools are numerous. Choosing the right one is a tedious job and is an important step. It is really important that the selected tool should be a good match for the company's problem at hand. Additionally it should address the problem efficiently and effectively. In this regard the selection should be made by evaluating it to the real requirements. The following section shows how this evaluation is performed in this thesis while selecting the test tool.

5.1 Tool Selection Criterions

The most vital part of this research was deciding on the right tool for automation. This first stage is treated as important because the whole automation effort has a direct impact on the selected tool. If the choice is right then the whole automation work will produce an effective result. In other case, the selection can adversely affect the company's profit factor.

Nowadays the ranges of choice of test automation tools available are wide. The true fact is that none of the available tools fully satisfies one's requirements. Considering this most of the quality assurance experts' view is to compare the tool with the project's requirements and select the one that is apt for one's requirements rather than selecting the most popular one.

In order to compare the tool with the project's requirements, evaluate the tool with respect to certain aspects or features. According to Plotytsia (2014), eleven steps are required to identify the right tool for test automation. This research follows these eleven steps, which are listed below, to evaluate the tools with respect to the project's requirements and thereby to choose the right tool. The evaluation of the tools based on the eleven steps is presented in Section 5.3.

11 Questions to ask for choosing the best automation testing tool:

Step1. Which mobile operating systems are supported?

This step checks if the tool supports the operating system (OS) in which the application runs. If it is a browser based web application then check if the tool could be used to execute the test cases on available browsers like FireFox, Chrome, IE etc. If the application is a mobile based one then consider the OS on which the test cases should be executed like Android, IOS, Windows Phone, BlackBerry etc.

Step2. Which type of mobile application is supported?

The second step is to ensure the tools support on the application type i.e. support for a web app, a native app or a hybrid app. Native apps which are installed through an application store are always present on the mobile device and could be accessed by tapping their icon. Web apps are mobile version of a web-site running on a mobile browser like Safari or Chrome. It looks exactly like a native app, but in reality it is not implemented. A hybrid app is a combination of native and web apps. The native features allow it to enter into the application store and could be installed from there. Similar to the web app, they are also rendered in a browser using HTML; however the browser is embedded within the app. In other words, one can see the hybrid apps as wrappers for an existing web page; thereby minimizing the development effort and making a solid presence in the app store.

Step3. Is the source code required?

Because of security reasons there are chances that the source code is unavailable for testing. This can affect the effectiveness of testing and so the tool should be selected accordingly.

Step4. Is application modification required?

In some cases the application available for testing requires some source code changes to be tested using a particular tool. So it is wise to consider this change while evaluating the tool.

Step5. In which way are the test scripts created?

Test scripts could be created via Recording/Playback approach which is considered as fast and easy way. Another one is programmatic approach which could utilize the capabilities provided by the programming language and also could exploit the powercoding feature. This second approach is not that fast as the first one but it is the most flexible and effective approach compared to the first.

Step6. Which programming language is utilized?

A tool that supports a language that is familiar to the available resources is essential to minimize the resources' learning curve and also to reuse the existing skill set.

Step7. How the object recognition is done?

Test script maintenance cost is an important factor to be considered while selecting a testing tool because application undergoes changes always before its release. Hence to minimize the impact of the application changes it is better to have unique object identification. Objects could be selected by name, id, class, XPATH, link test, CSS selector and JavaScript. Also it is better if the tool supports object library for mapping objects as they can be easy to update and manage.

Step8. Does the tool support data driven inputs?

Ability to connect to any of the data source is another beneficial feature that the tool should possess, especially if the automation uses keyword-driven or data-driven frameworks. Common data sources such as spread sheets, .csv files, XML files and database storages could be connected via appropriate drivers.

Step9. How detailed is the result logging?

Reporting mechanism is another factor to consider while selecting a tool. If a test passes then the log need not be too descriptive, just the duration of run and the running environment information could suffice. However, a failed test log needs to be descriptive enough to point out the exact point where the script fails. Another added feature could be to have a screenshot of the moment of failure. In order to share with stakeholders and other partners it will be an added plus if the log report could be exported to other formats and could be customized.

Step10. Which options of the integration with other tools are available?

The selected tool should be integrated well with the existing testing environment so that the team could manage the whole application life cycle effortlessly. The tool should support the testing infrastructure components like CI, CM, IDE, revision tool, test frameworks, test management, report logging, bug tracking etc.

Step11. What is the pricing model?

Price of the tool is another concerning factor for a company. Tools are available in the market for free to use or with a fee. Free open source tool should be checked for its evolution stability and also for its fastness in supporting the technological changes. Fee of the tool includes license fee, add-ons fee, upgrade fee, training fee and support fee. License can also be of various types like 1) Node-Locked User License that allows one instance of the tool to run on a single computer in company’s network 2) Concurrent Floating User License that could be shared with different machines, but could run only one instance at a time 3) Run Time License enables to use the tool on different machines at the same time. So consider buying the tool after taking into account the ROI of the automation effort.

5.2 Requirements of Metinfo for Automation Test Tool

This section describes the most important requirements for selecting the best tool for testing. The requirements are collected by arranging a meeting with product owner. These requirements should be analyzed well for deciding the tool.

The first requirement was to select a tool that could be integrated well with the existing test environment. The tool should at least support Jenkins for continuous integration. Another requirement was to select an open source tool. The company is a big organization and getting funding for license is a long process and will take long time to get approval. Hence according to the company, the option to select a licensed tool should be avoided as far as possible.

The third requirement was that the tool should support Android and iOS operating systems as most of its customers are using those OS. The tool is required to handle web app as well as native app. Most of the end users use web app for getting the information. But there are certain cases when users may need to record some of the data while they are offline. In those cases there are some native apps which they are using internally. These need to be tested in the future.

Another important factor to consider is that because of security reasons and long time for getting approval, it is not possible to get the source code. The tool should work without providing the source code. Also it is beneficial that the tool should support all of the programming languages. The skill set of the available resources of the company is low and so they prefer to have a tool that supports almost every language.

5.3 Tool Evaluation Based on Criterions

This section evaluates some of the selected commercial as well as open source automation tools based on the eleven questions of Plotytsia (2014) and analyzing that with the real requirements of Metinfo.

The research considered ten of the most popular mobile automation tools. These ten are listed below:

- Calabash - Supports both Android and iOS (Calabash 2015)
- Appium - Supports both Android and iOS (Appium 2015)
- MonkeyTalk - Supports both Android and iOS (CloudMonkey 2015)
- Sikuli – It uses screen captures as image recognition for testing the GUI. Though it employs visual technology it is less powerful than other competitors. It could be used in combination with other tools to make an effective testing. (Sikuli 2015)
- Robotium – Intended for Android test automation (Robotium 2015)
- Selendroid – For Android native or hybrid or mobile web apps (Selendroid 2015)
- Frank – Intended for iOS applications (Frank 2015)
- KIF - KeepItFunctional, targeted for only iOS (KIF 2015)
- eggPlant - Supports both Android and iOS (TestPlant 2015)
- TouchTest - Supports both Android and iOS (SOASTA 2015)

Among the ten tools Calabash, Appium, MonkeyTalk, eggPlant and TouchTest are selected for detailed study and are presented in Sections 5.3.1, 5.3.2, 5.3.3, 5.3.4 and

5.3.5. The reason for rejecting other tools is their lack of support for both Android and iOS. Though Sikuli uses image captures for testing it is considered less powerful than the selected five tools. The following section shows a detailed study of the selected five tools based on the eleven questions prepared by Plotytsia (2014).

5.3.1 Calabash

Calabash is an automated test framework for Android and iOS. It is cross-platform with tests written in Cucumber that follows a Behavior Driven Development (BDD) methodology. BDD approach allows to express the requirement specification using Gherkin; a set of easy-to-understand natural language grammar rules. This kind of language syntax enables Calabash to express the domain-specific features in a business readable format and there by non-technical staff and business experts could ensure the correctness of the specifications, even before the real implementation happens. Figure 17 illustrates BDD requirement specification in Calabash.

```
Feature: Credit card validation.
Credit card numbers must be exactly 16 characters.

Scenario: Credit card number is too short
   Given I use the native keyboard to enter "123456" into text field number 1
   And I touch the "Validate" button
   Then I see the text "Credit card number is too short."

Scenario: Credit card number is too long
   Given I try to validate a credit card number that is 17 characters long
   Then I should see the error message "Credit card number is too long."
```

Figure 17: BDD sample in Calabash (Calabash 2015)

Step1. Which mobile operating systems are supported?

Calabash is a great cross-platform framework for test automation and is used to test Android and iOS native and hybrid apps.

Step2. Which type of mobile application is supported?
Mainly Calabash could be used to test native apps. However, hybrid apps are also supported by means of using some libraries. In this case the application web elements are identified by using JavaScript or Ruby.

Step3. Is the source code required?
Source code is not required for test script creation and execution. This advantage applies for both iOS and Android applications.

Step4. Is application modification required?
Though source code is not a necessity to create and execute tests, it is required to modify the source code. Thus a separate version of the application is needed with the below changes, which is truly a negative-side of Calabash.

The modifications needed for Android and iOS also varies a little bit. In case of Android and iOS, Ruby and Cucumber libraries need to be included in the project and make entries to the path variables. In addition to this, iOS applications should embed an HTTP server in it by connecting with the framework: calabash.framework. This kind of duplication of production version is an extra work load with Calabash.

Step5. In which way are the test scripts created?
Record or Playback feature is supported on iOS6 and lower versions. From iOS7 onwards, Apple has removed this feature. However, gestures like pinch, pan and swipe could be recorded in iOS7 by bridging with the UIAutomation frameworks or scripting language using the uia_* set of functions.

In order to record touches and other gestures in iOS < 7, one can use the calabash-ios gem from the irb (Interactive Ruby Shell). First one has to record the touch event sequences as one carries out on a real device or on a simulator. While launching the application on a real device or on simulator Calabash libraries are included. Secondly, playback the recorded sequence later as part of a step in a Cucumber feature. The record_begin, record_end and playback are the API functions used for record and playback. These methods are not yet implemented in calabash-android.

Data-driven testing is not supported in Calabash by default. However, Ruby which is playing behind the screens makes data-driven testing possible by including some Ruby codes to calabash-steps.rb. In this case an editor can be used to create the test scripts. On the other hand, keyword-driven testing is not yet fulfilled by Calabash and therefore script-less testing is not possible so far with Calabash.

Step6. Which programming language is utilized?

Calabash uses Gherkin as its programming language which could be understandable by Cucumber. It is a business readable, domain specific language with a set of easy-to-understand natural language grammar rules. With this syntax software's behavior is described well without doing the real implementation. Also with this language the specifications are written in natural language making it understandable by non-technical people and also business experts. In order to test a feature, many scenarios will be identified and each scenario consists of many test steps which are written in Gherkin. These steps are programmed in Ruby and later the test steps are embedded into the Gherkin code.

Step7. How is the object recognition done?

Direct object recognition is not possible in Calabash. However, to interact and inspect web views four API's are available: XPATH, CSS, JavaScript and marked (free text matching). The query and gesture API's are supported on UIWebViews and WKWebViews. Figure 18 illustrates Calabash's object recognition.

```
query("webView css:'a'").first
touch("webView css:'a'")
enter_text("webView css:'input.login'", "run")
```

Figure 18: Calabash's object recognition (Calabash wiki 2015)

The query function lets to look into a web view. The elements that are visible in the screen could use the touch method in the Calabash API. The method enter_text allows entering a text in a web view.

Step8. Does the tool support data driven inputs?

Though data-driven testing is not supported in Calabash by default, Ruby enables it to perform data-driven testing. Ruby has a CSV library which allows Calabash to connect to external data source like .xls or .csv files.

Step9. How detailed is the result logging?
IDE terminal window outputs the test results showing the whole scenario under test and if the scenario as whole is passed or not and also how many steps passed successfully. In addition to this console output, user can specify output format like html, json etc. In order to get a screenshot of the error log, it is possible to provide the option screenshot_embed with the filename and the path where it should be saved.

Step10. Which options of the integration with other tools are available?
It support Continuous Integration and Jenkins could be connected to it for CI. Also version controlling like GitHub can be integrated. Calabash tests can be created using any of the text editors. Additionally, it supports IDE’s like IntelliJ, Eclipse, RubyMine etc.

Step11. What is the pricing model?
As Calabash is an open source tool, it is freely available for testing. It is available in GitHub to download freely. Though its use is free Xamarin provides several commercial services in the name of Calabash. They offer trainings and support services which are available for a fee.

5.3.2 Appium

Appium is an open source tool powered by Sauce Labs Inc., the leading provider of cloud-based test automation services for mobile and web applications. It is cross-platform allowing writing tests for multiple platforms i.e. iOS and Android.

Step1. Which mobile operating systems are supported?
Android, iOS and FirefoxOS are the supported platforms.

Step2. Which type of mobile application is supported?
Native apps, web apps and hybrid apps are supported by Appium.

Step3. Is the source code required?
No source code access is required.

Step4. Is application modification required?
No application changes are required.

Step5. In which way are the test scripts created?

Test script creation is performed in two ways. Using the GUI of appium.app it is possible to capture actions and later export them to other programming languages. It is also possible to modify the script and put it back to appium.app and play them back in real device or on emulator or on simulator. Another way is to create it using manually writing the scripts using any editor. Tests can be imported to appium.app and executed. Execution of the script could also be done using the terminal.

Step6. Which programming language is utilized?

All common programming languages like Java, JavaScript, Ruby, Python etc. and all frameworks are supported as Appium is based on Selenium Web driver.

Step7. How the object recognition is done?

True object recognition is provided by appium. iOS objects can be identified by three ways and Android objects can be identified by four ways: using element type, accessibility label, hierarchy and Android ID.

Step8. Does the tool support data driven inputs?

If the programming language used supports data driven inputs, then Appium also supports.

Step9. How detailed is the result logging?

Debug and error messages are displayed in the terminal. In case of appium, those are displayed in the tools terminal. For detailed log like the test data, steps, error messages with screenshots etc. Sauce Lab offers the feature for those having a Sauce Lab account.

Step10. Which options of the integration with other tools are available?

Appium can be integrated well with Selenium and also with the development IDE’s used by the developer.

Step11. What is the pricing model?

It is an open source tool and so can be used freely. However no support is offered by Sauce Lab Inc. and only support is in the form of GitHub project and also from Google groups.

5.3.3 MonkeyTalk

MonkeyTalk also supports both iOS and Android platforms. This was formerly known as FoneMonkey. CloudMonkey Mobile released MonkeyTalk and they are available both as open source as well as subscription based commercial solution. The commercial version, MonkeyTalk Pro, has some additional useful features.

Step1. Which mobile operating systems are supported?
Both iOS and Android is supported by MonkeyTalk.

Step2. Which type of mobile application is supported?
All application types – Web, Mobile and Hybrid – are supported by MonkeyTalk.

Step3. Is the source code required?
Source code is not required for creating test scripts.

Step4. Is application modification required?
MonkeyTalk IDE which is an Eclipse IDE requires MonkeyTalk agent to be embedded in the application. It is essential for the communication between the IDE and the application.

Step5. In which way are the test scripts created?
Creation of test script is done in two ways: either using the IDE's record feature or manually using an editor. If script is created manually it must be saved as .mt so that it can be imported it to the IDE and executed. Modification is possible with IDE or with editor. Editor modified ones could then be imported to IDE for execution. Also the modified scripts could be run in Ant Runner or in Java Runner.

Step6. Which programming language is utilized?
As of now, MonkeyTalk test scripts can be converted to JavaScript. No other language bindings are supported.

Step7. How the object recognition is done?
Similar to appium, true object recognition is provided by MonkeyTalk. iOS objects can be identified by three ways and Android objects can be identified by four ways: using element type, accessibility label, hierarchy and Android ID.

Step8. Does the tool support data driven inputs?
Data-driven testing is supported with spreadsheet files.

Step9. How detailed is the result logging?
Result logging can be done in various formats. HTML, XML, xUnit reporting is supported with detailed test results like screenshots and step-by-step logging.

Step10. Which options of the integration with other tools are available?
Integration is possible with other tools like xUnit standard and Jenkins.

Step11. What is the pricing model?
It is available for free. However, the features available are less and also no technical support from CloudMonkey Mobile. However, these features are available with the professional edition.

5.3.4 eggPlant

One of the most popular tools is eggPlant. It introduces a unique technology for testing by performing the test with what user sees and acts on the application. This technology of image recognition allows using just the GUI to create the test cases without any need of source code.

Step1. Which mobile operating systems are supported?
This tool supports iOS, Android, Windows Phone and Blackberry.

Step2. Which type of mobile application is supported?
All application types – Web, Mobile and Hybrid – are supported by eggPlant.

Step3. Is the source code required?
As eggPlant works using the GUI, it does not require the source code.

Step4. Is application modification required?
It just requires an additional program to be installed in the computer and on the device for their communication. No other source code or application modification is required.

Step5. In which way are the test scripts created?
Scripts are created in two ways. The first way is to use an IDE for capturing the actions. The tapped images and expected actions are saved and these captured images serves as the objects. The second way is to write the scripts manually and run on simulator or emulator or device.

Step6. Which programming language is utilized?
SenseTalk, a member of the xTalk scripting language, is the programming language used. Though it is not that powerful like JavaScript or Python, it is an easy to use English-like language that allows even any non-programmer to write the scripts.

Step7. How the object recognition is done?
Image recognition is the technology used and so instead of object parameters from source code the captured images serves as the object. So true object recognition is not supported. Also hidden objects cannot be identified. But it works successfully with complex graphical cases.

Step8. Does the tool support data driven inputs?
XML, CSV, txt files are supported to provide data sets.

Step9. How detailed is the result logging?
The IDE provides an extra window named report suite where the test result could be viewed. It shows the test case, test step, any errors, execution time, screenshots etc. The report could then be exported as HTML or log file.

Step10. Which options of the integration with other tools are available?
The tools that could be integrated with eggPlant are IBM Rational Quality Manager (IBM RQM), Jenkins, HP Application Lifecycle Management (HP ALM).

Step11. What is the pricing model?
Its license cost depends on whether it is used by a single person or by a team. Some support in the form of information material and webinars are included with this license. However, coaching and training costs additional charge.

5.3.5 TouchTest

SOASTA is the provider of TouchTest.

Step1. Which mobile operating systems are supported?
Both iOS and Android platforms are supported.

Step2. Which type of mobile application is supported?
All application types – Web, Mobile and Hybrid – are supported by TouchTest.

Step3. Is the source code required?
No source code is required to create or run the test scripts.

Step4. Is application modification required?
TouchTest libraries should be added to the application and can be done using SOASTA's tool MakeAppTouchTestable (MATT).

Step5. In which way are the test scripts created?
Test scripts creation is performed in two ways. One is precision gesture recording and the other is using manual creation using an IDE or editor.

Step6. Which programming language is utilized?
JavaScript is the language supported as the recorded data need to be translated into App Actions and later played back repetitively.

Step7. How the object recognition is done?
True object recognition is supported. Objects are identified with their ids or developer set parameters.

Step8. Does the tool support data driven inputs?
XML and CSV files are supported to supply data sets.

Step9. How detailed is the result logging?
TouchTest IDE provides a result view that shows the test case, test step, error or success messages and screenshots. This result can then be exported to CSV or XML file.

Step10. Which options of the integration with other tools are available?
Jenkins could be integrated with TouchTest.

Step11. What is the pricing model?
It is a commercial solution.

5.4 Tool Comparison

The general requirements of the company are met by the five tools analyzed. However, to select a specific tool for the research it is essential to compare them with each other. The comparison between the five tools are presented in Table 8.

Table 8: Tool comparison

Criteria	**Appium**	**Calabash**	**MonkeyTalk**	**eggPlant**	**TouchTest**
iOS and Android support	✓	✓	✓	✓	✓
Native, Web or Hybrid App support	Native, Web and Hybrid	Native and Hybrid	Native, Web and Hybrid	Native, Web and Hybrid	Native, Web and Hybrid
Need of Source code	No	No	No	No	No
Need of App modification	No	Yes	Yes	No	Yes
Creation of Test script	Record; Manual script via editor	Record; Manual script via editor	Record; Manual script via editor or IDE	Record; Manual script via editor or IDE	Record; Manual script via editor or IDE
Programming language	All common languages can be used	Gherkin® (additional languages are possible)	MonkeyTalk, JavaScript©	SenseTalk®	JavaScript©

True Object recognition support	✓	✗	✓	✗	✓
Data-driven input support	✓	✓	✓	✓	✓
Result logging	IDE's Output console; Report file on Sauce Lab website	IDE's Output console; Test report as html, json etc.	HTML document; XML export; xUnit® export	IDE report view; HTML document	IDE report view; XML and CSV export
Integration with other tools	Selenium® and development IDE	Jenkins® and GitHub	Jenkins® and xUnit® frameworks	Jenkins® , IBM RQM® , HP ALM®	Jenkins®
Pricing	Free	Free	Free	Commercial	Commercial
Criteria	**Appium**	**Calabash**	**MonkeyTalk**	**eggPlant**	**TouchTest**

Table 8 shows that all five tools support iOS and Android platforms. For the three type of application support Calabash did not support. Though source code is not a necessity for testing, for test script creation source code or application modification is required except for Appium and eggPlant. All tools have record/playback and manual script writing feature. Coming to programming language support just Appium supports all languages which are a favorable factor for Appium.

True object recognition support is provided by Appium, MonkeyTalk and TouchTest. Though all tools support data sets from external files, only some of the file types are supported. But many of them support spreadsheets. Additionally, all tools have some kind of report logging either IDE's output console or exporting to some other file formats. Almost all tools support integration with Jenkins also. A final comparison criterion is the pricing model. The first three tools – Appium, Calabash, and MonkeyTalk – are available for free and last two tools – eggPlant, TouchTest – are commercial versions. For the open source solutions limited support is provided and with commercial versions they charge additional amount for extended support.

5.5 Selected Tool

Based on the analysis done the right tool and framework for this research work is identified by comparing the features provided by the tool with the requirement of the company. First of all the company needs a tool that supports iOS and Android operating systems and all tools compared satisfy this requirement.

Calabash does not support mobile web apps and programming language utilized is Gherkin. Also application modification is required for script writing and also it does not support true object recognition. These limitations set Calabash away from the list.

MonkeyTalk and TouchTest also require the source code or application to be modified for writing the scripts. This is a big disadvantage because the company specification says that the test automation has to be done without accessing the source code. Also the programming language support is provided just for JavaScript and MonkeyTalk. Though language support is not a great factor, company prefers a tool having support for more languages so that company can use existing resources for script creation or management. For these reasons, MonkeyTalk and TouchTest are also out from the selection list.

Coming to eggPlant, it almost matches with the company’s requirements. However, certain factors are not in favor of eggPlant. Among them the main one is its lack of having true object recognition feature and its image recognition technology for identifying the objects. Another is its programming language as SenseTalk. Though SenseTalk is easy to understand yet it is a new language for many of the existing resources. As these two drawbacks are considered, its commercial pricing model also put this to the back side.

Now Appium comes forward in the list and it has many supporting features with respect to the requirement. Comparing with other tools it supports all programming languages and also true object recognition is provided. It is an open source tool that does not require any source code for script writing and also no code modification is needed. Thus Appium was selected as the right tool for this research work.

6 Continuous Integration Environment

Agile methodologies are based on short iterations and short release cycles. So to ensure the quality of the delivered software and to make efficient testing effort test automation is introduced. However, to run the automated tests repeatedly whenever a code changes is possible only with some software tool. Otherwise it is considered as an overhead to trigger the automated tests manually whenever a change happens. The process that helps to run the test continuously is called continuous integration. CI executes the automated tests after any commits. Developers will be sent the test results soon after the commit, thereby errors are identified easily. So it is important to integrate automated tests with a continuous integration environment. The tools needed to implement a continuous integration environment are:

- Test Frameworks
- Continuous Build Systems / Continuous integration server
- Code Repositories / Artifact Repositories

The listed tools form the basic infrastructure for the continuous integration environment. Each of these tools are described in detail in the following Sections.

6.1 Selected Test Framework

To implement continuous integration, it is essential to have a testing framework for executing all the test scripts. This testing framework helps to abstract the test automation tool and provide a single place to execute the scripts. That is, instead of doing any specific testing activity this testing framework acts as a front end for libraries like Selenium2Library and AppiumLibrary. In this regard, the next step is to select the testing framework for implementing the continuous integration.

The conditions for selecting the right test framework depend upon the project requirements. For this research the conditions that should be satisfied by the test framework are:

- It should have some support to make sure its longevity and continued development
- It should support a number of testing tools
- It should be extensible to add various test tools.

- It should integrate well with continuous integration environment
- It should support the languages Java and Python because the existing resources already know those two languages
- It should be available open source
- It should provide good documentation

After careful analysis the framework that satisfies all of the specified requirements was Robot Framework. It is an open source tool sponsored by Nokia Siemens Networks and is released under the license of Apache 2.0. It continuous to be free and ensures some longevity.

It supports Java and Python languages and supports integration with continuous integration environment. It is simple and easy to understand allowing even non-technical resources to maintain the test scripts. It supports various testing tools like Appium, Selenium etc.

Robot framework has other benefits, too. It is a generic test automation framework that uses keyword-driven testing. Its tabular test data syntax makes it a simple and easy to use framework. Because the keywords are described in real language even non-technical persons can handle it. It has already some defined libraries like String, Collection etc. However these libraries could be extended for more testing capabilities using Python and Java. Additionally the keywords for testing could be created even before the UI is implemented.

6.2 Selected Continuous Integration Server

Continuous integration automates the build and test case execution and thereby accelerates the software development process. It reduces the risk of bug detection at the later stages of the release. Its philosophy of incremental test makes it easier to detect defects and fix it early whenever it is introduced. This lowers the risk threshold and helps in faster delivery of quality software enforcing agile development.

The company already has Jenkins as the continuous integration server for some other projects. Its flexibility and core functionality fits it in a variety of environments. It is branched from Hudson and is Java based which is under development since 2005. It supports over 400 plugins for reporting, notifications, testing etc. Jenkins grabbed lot of awards from 2008 and is so popular with its large number of users. Its support community is large. It is easy to write plugins and has a great documentation.

It integrates well with different version control systems. It can generate useful test reports. It provides deployment directly to test environments or to production. It can also notify stakeholders about the current build status alerting them of any failed test cases. The benefits and features of Jenkins are numerous. However, the above capabilities are best matched with the requirements of the company. In this regard, Jenkins was selected as the continuous integration server.

6.3 Selected Artefact Repository

The existing project is hosted at GitHub. It is a distributed version control system. It is one of the most popular open source systems for versioning. It has proper excellent documentation and also the number of users is large. It is purely web-based promoting collaboration among team members.

GitHub has other supporting features as well. Its Pull Request (PR) mechanism initiates discussion with other members and also enables to share or comment on the project’s artifacts. The shared repository model and the Fork & Pull model allow anyone to contribute to the project.

GitHub also allows users to follow an interested project and also to broadcast their activities to followers. New projects could be searched by the explore feature and Gists feature helps to share snippets. The GitHub’s trasparency feature benefits the community’s collaboration. Considering all these factors, this study selected GitHub for the Artifact Repository.

6.4 Continuous Integration Pipeline

Originally Continuous Integration means that running the automated tests for every code change. Continuous integration pipeline represents the flow from developer making the code change till the testing of the system change and test report generation. This section shows this flow and how the selected tool, test framework, integration server and artifact repository fit together into the continuous integration testing system for making the continuous integration pipeline. Figure 19 illustrates the continuous integration pipeline.

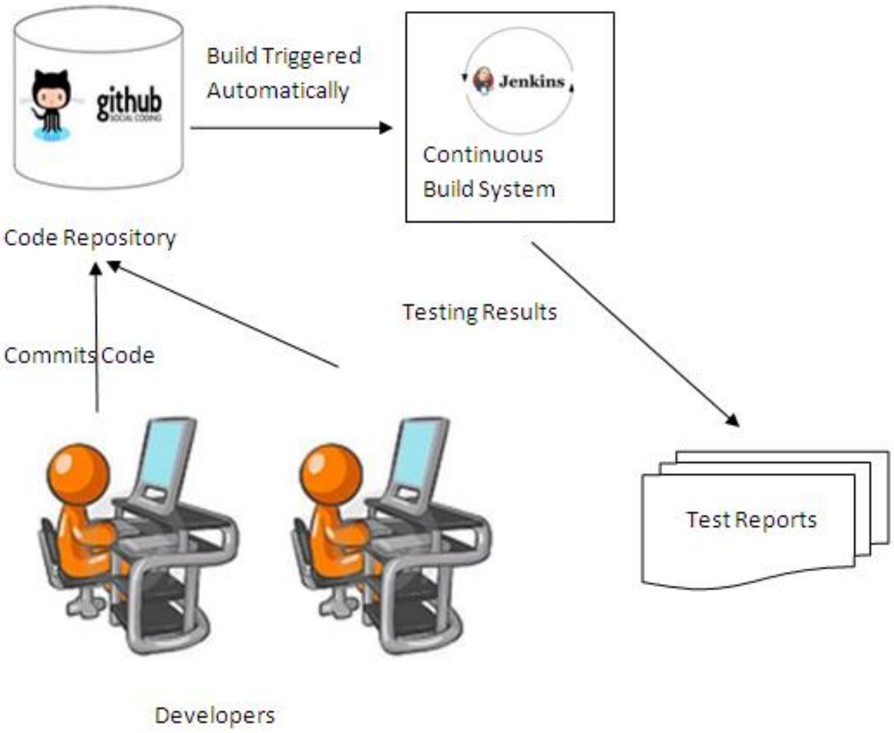


Figure 19: Continuous Integration Pipeline

Figure 19 above shows the continuous integration pipeline for the project. From a high level, the pipeline consists of the below specified steps:

1. *Code Commit*: The first step to initiate the flow is to make some changes to the code by the developer. Once the change is done, developer commits the code to a central source code repository. In this project it is GitHub.
2. *Build*: The Continuous Build System which is Jenkins in this project, continuously polls GitHub for any code changes. Whenever it identifies a change in the source code because of the recent commit by the developer it initiates a new build with the changed system. For that, the change is checked out from the repository and build starts.
3. *Automated Tests*: Once the build finishes, the automated test scripts are pulled from the Artifact Repository, in this project case it is GitHub, and start to execute all the test cases. This makes sure that nothing breaks because of the new change.

4. *Test Reports*: Once the script execution is finished, test reports are generated with the execution status. Any failure is reported back to developer for further fixes. Also notifications are sent to stakeholders.

On a high level the system architecture is shown in Figure 20:

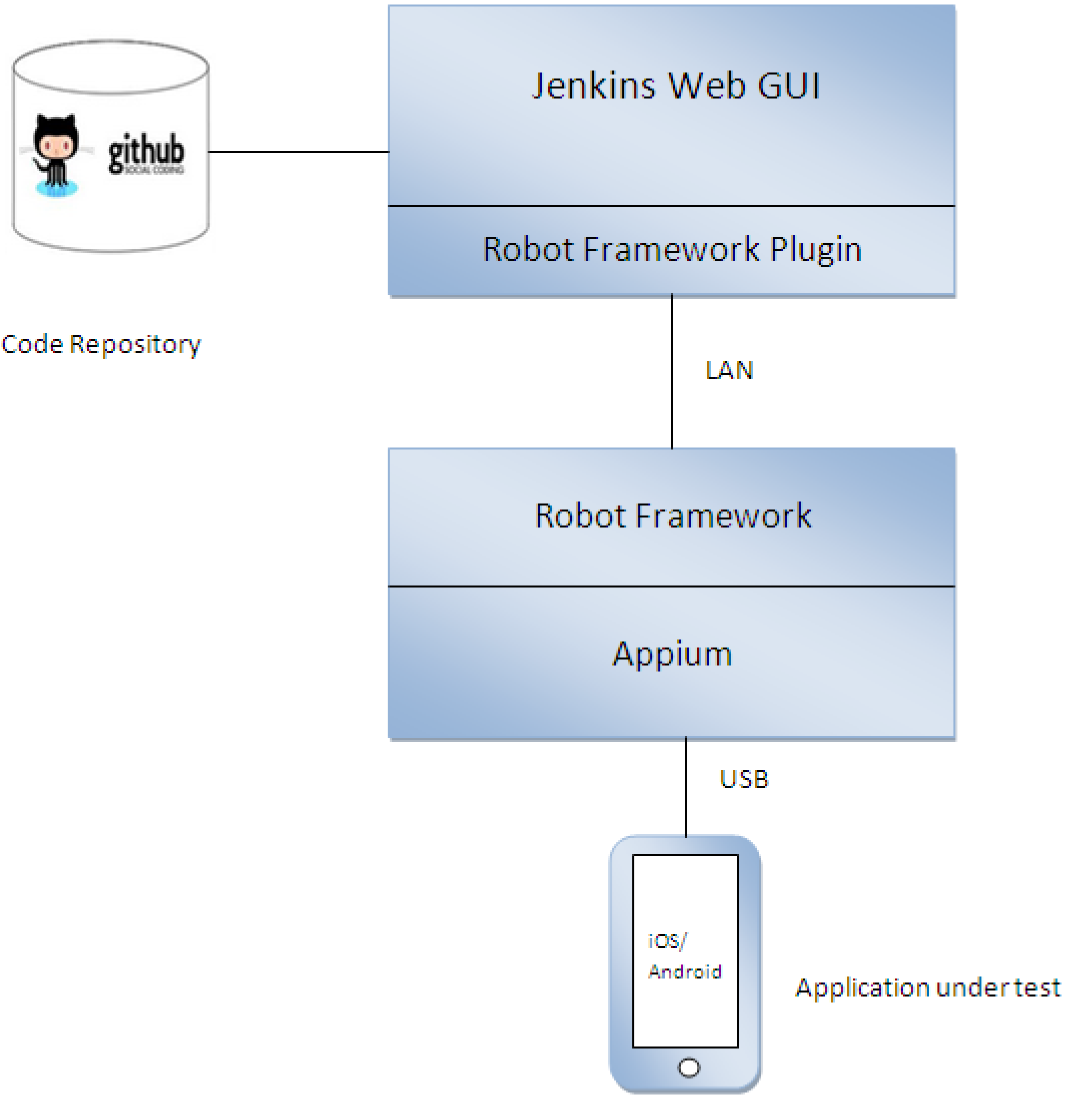


Figure 20: System architecture

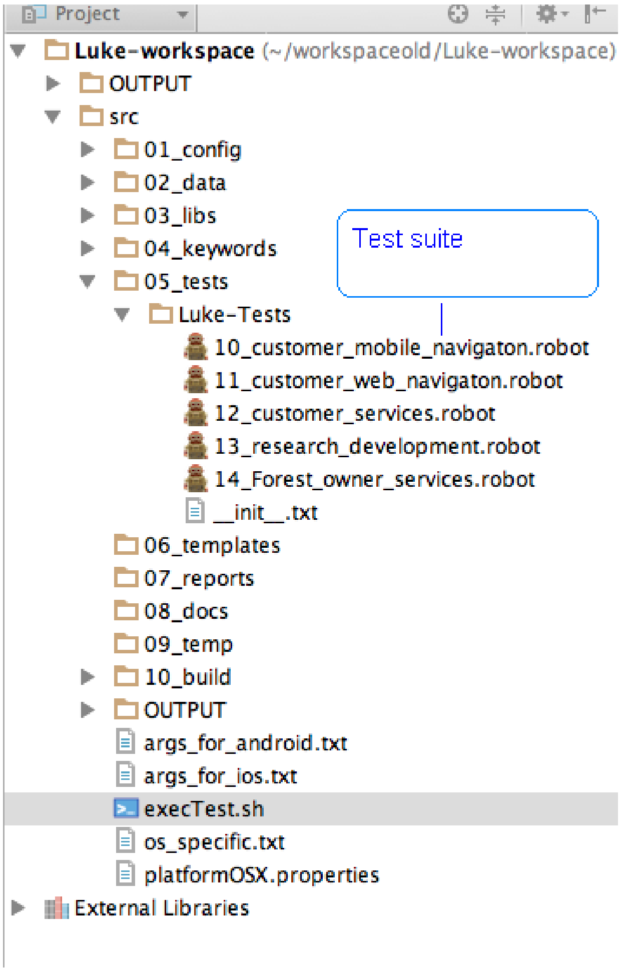
The above figure shows the architecture of the automated system. It gives an idea about how the selected tool, framework and continuous integration server are connected together. Git is the version control tool where the source code and test files are stored. Jenkins is the continuous integration server. Jenkins Git plugin pushes a job to Jenkins whenever source code change occurs in Git. Jenkins Robot Framework plugin starts the test execution and fetches test results back to Jenkins. Robot Framework is the test framework and is the test automation execution engine. This framework does not have any knowledge about the target under test and test libraries handle this interaction. Here Appium is the test interface or test library used. Target device where application under test running is on iOS or Android.

7 Implementation of Test Automation

This section details the steps performed to create the test cases and its execution.

7.1 Metinfo Test Scripts Workspace Structure

The workspace structure for the Metinfo test scripts is shown below in Figure 21:

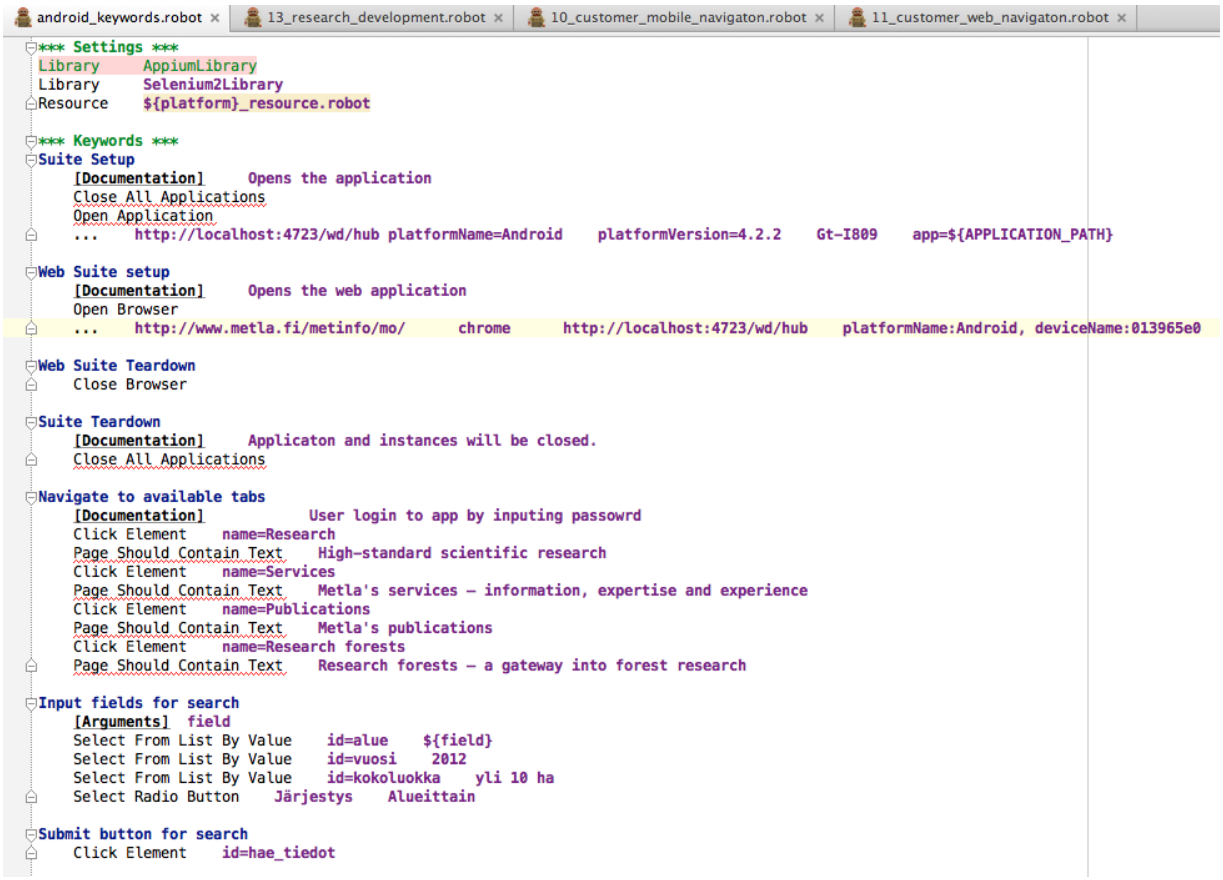


```
▼ Luke-workspace (~/workspaceold/Luke-workspace)
  ▶ OUTPUT
  ▼ src
    ▶ 01_config
    ▶ 02_data
    ▶ 03_libs
    ▶ 04_keywords
    ▼ 05_tests
      ▼ Luke-Tests
          10_customer_mobile_navigaton.robot
          11_customer_web_navigaton.robot
          12_customer_services.robot
          13_research_development.robot
          14_Forest_owner_services.robot
          __init__.txt
      06_templates
      07_reports
      08_docs
      09_temp
    ▶ 10_build
    ▶ OUTPUT
      args_for_android.txt
      args_for_ios.txt
      execTest.sh
      os_specific.txt
      platformOSX.properties
▶ External Libraries
```

Figure 21: Project structure

The root folder represents the workspace name and is given as Luke-workspace. All other files like test suites, configuration files, keywords, reports etc. are contained in the src folder.

- 01_config folder has the configuration files.
- 02_data contains all the data sets used for data-driven testing as well as a python library for accessing these data sets.
- 03_libs folder is for extending the Appium library as well as selenium library using python.
- 04_keywords folder stores all the keywords used for testing. Figure 22 presents the android keywords file.

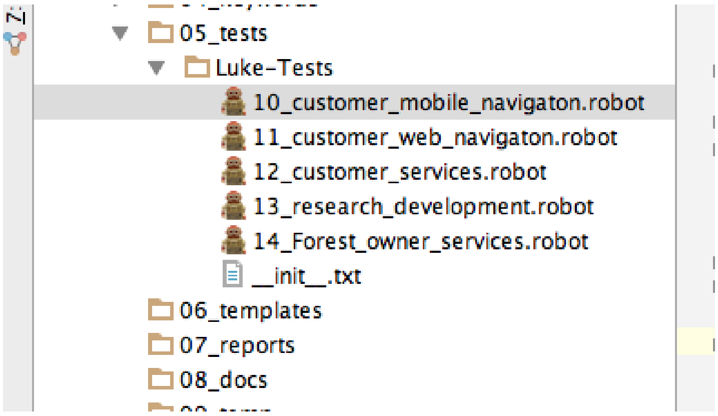

android_keywords.robot
13_research_development.robot
10_customer_mobile_navigaton.robot
11_customer_web_navigaton.robot

```
*** Settings ***
Library      AppiumLibrary
Library      Selenium2Library
Resource     ${platform}_resource.robot

*** Keywords ***
Suite Setup
    [Documentation]     Opens the application
    Close All Applications
    Open Application
    ...     http://localhost:4723/wd/hub platformName=Android    platformVersion=4.2.2    Gt-I809    app=${APPLICATION_PATH}

Web Suite setup
    [Documentation]     Opens the web application
    Open Browser
    ...     http://www.metla.fi/metinfo/mo/      chrome      http://localhost:4723/wd/hub    platformName:Android, deviceName:013965e0

Web Suite Teardown
    Close Browser

Suite Teardown
    [Documentation]     Applicaton and instances will be closed.
    Close All Applications

Navigate to available tabs
    [Documentation]            User login to app by inputing passowrd
    Click Element    name=Research
    Page Should Contain Text    High-standard scientific research
    Click Element    name=Services
    Page Should Contain Text    Metla's services – information, expertise and experience
    Click Element    name=Publications
    Page Should Contain Text    Metla's publications
    Click Element    name=Research forests
    Page Should Contain Text    Research forests – a gateway into forest research

Input fields for search
    [Arguments]  field
    Select From List By Value    id=alue    ${field}
    Select From List By Value    id=vuosi    2012
    Select From List By Value    id=kokoluokka    yli 10 ha
    Select Radio Button    Järjestys    Alueittain

Submit button for search
    Click Element    id=hae_tiedot
```

Figure 22: android_keywords file

The 04_keywords folder contains android_keywords.robot and ios_keywords.robot file. They are the defined keywords used for testing. Figure 22 shows the keywords for testing in android.

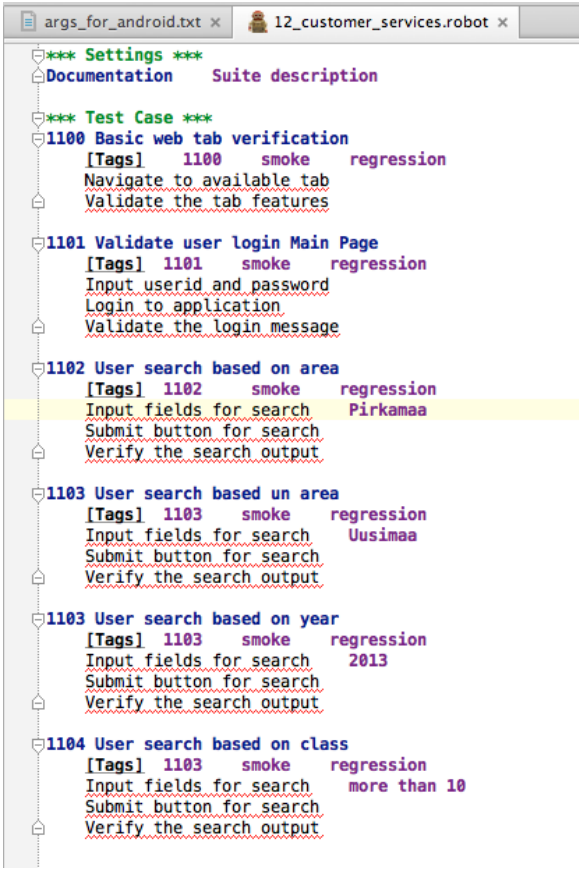
- 05_tests folder includes all the test suites. Figure 23 presents the test suites used for testing.



```
05_tests
    Luke-Tests
        10_customer_mobile_navigaton.robot
        11_customer_web_navigaton.robot
        12_customer_services.robot
        13_research_development.robot
        14_Forest_owner_services.robot
        __init__.txt
06_templates
07_reports
08_docs
```

Figure 23: Test suites

Figure 23 shows the test suites created like 13_research_development.robot, 14_Forest_owner_services.robot etc. Inside these test suites there are many test cases. These test cases are presented in Figure 24.

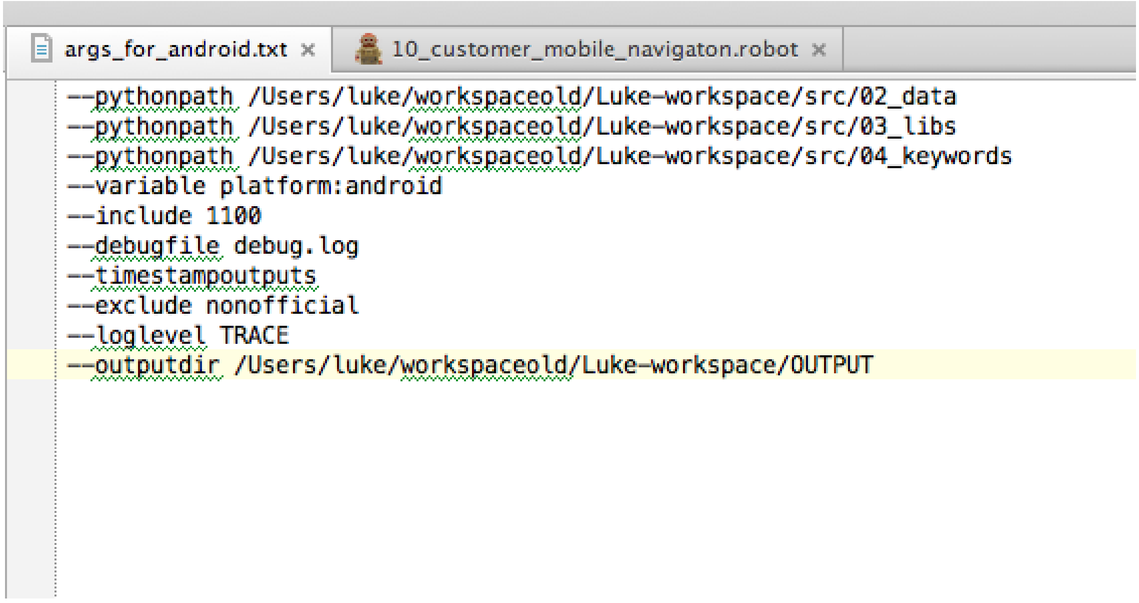


```
*** Settings ***
Documentation    Suite description

*** Test Case ***
1100 Basic web tab verification
    [Tags]    1100     smoke     regression
    Navigate to available tab
    Validate the tab features

1101 Validate user login Main Page
    [Tags]  1101     smoke     regression
    Input userid and password
    Login to application
    Validate the login message

1102 User search based on area
    [Tags]  1102      smoke     regression
    Input fields for search    Pirkamaa
    Submit button for search
    Verify the search output

1103 User search based un area
    [Tags]  1103     smoke     regression
    Input fields for search    Uusimaa
    Submit button for search
    Verify the search output

1103 User search based on year
    [Tags]  1103     smoke     regression
    Input fields for search    2013
    Submit button for search
    Verify the search output

1104 User search based on class
    [Tags]  1103     smoke     regression
    Input fields for search    more than 10
    Submit button for search
    Verify the search output
```

Figure 24: Test cases

Figure 24 above shows some of the test cases from one of the test suite named 12_customer_services.robot. Those test cases are shown in blue color. Each of the test case has several associated steps and a tag name.

- 06_templates for creating templates
- 07_reports contains the test report
- 08_docs for storing documentation of the keywords and test cases
- 09_temp for storing any temporary files
- 10_build for storing build files for local automation development purpose
- The src folder also contains some argument files. One of the argument file is presented in Figure 25.

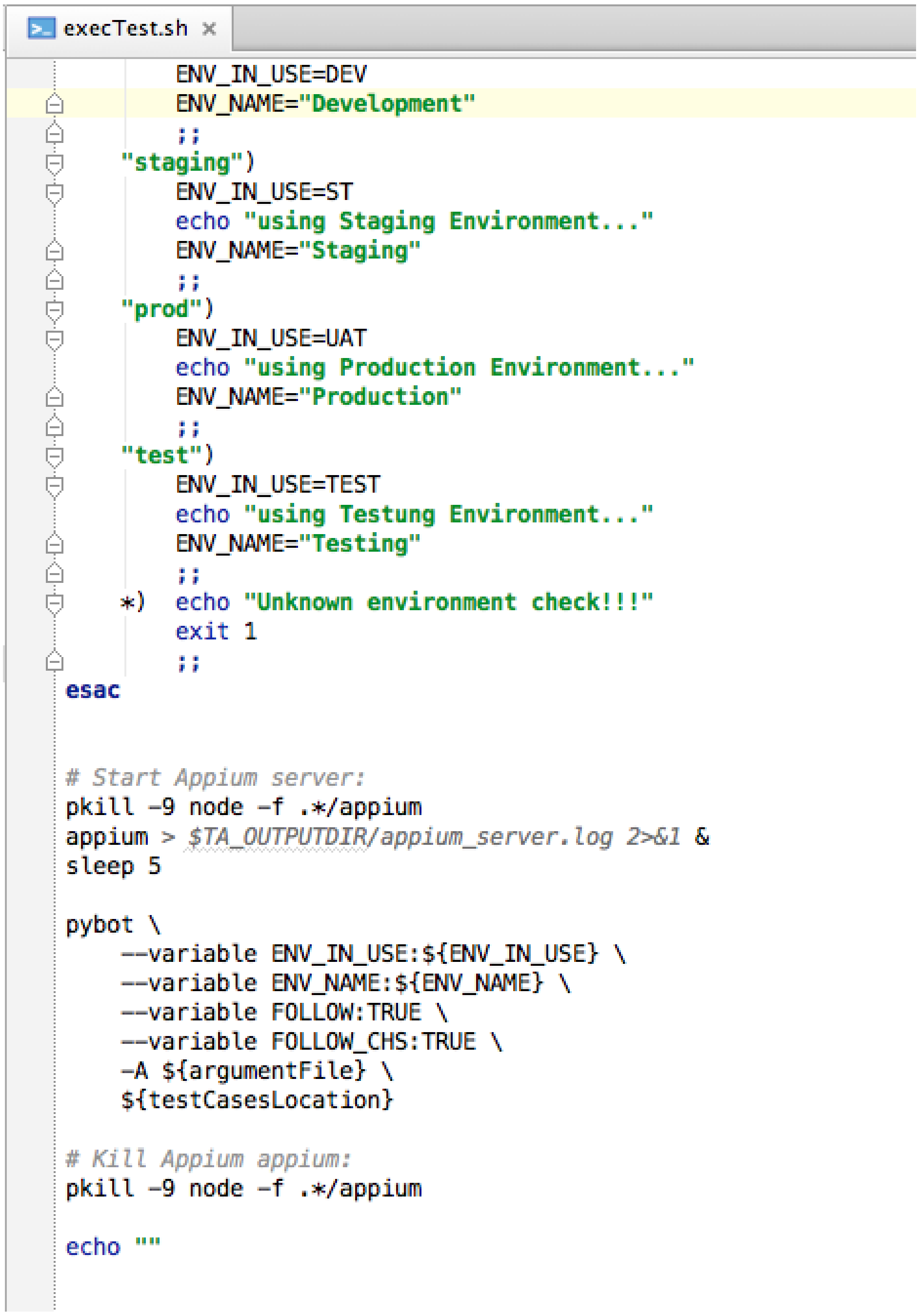


```
--pythonpath /Users/luke/workspaceold/Luke-workspace/src/02_data
--pythonpath /Users/luke/workspaceold/Luke-workspace/src/03_libs
--pythonpath /Users/luke/workspaceold/Luke-workspace/src/04_keywords
--variable platform:android
--include 1100
--debugfile debug.log
--timestampoutputs
--exclude nonofficial
--loglevel TRACE
--outputdir /Users/luke/workspaceold/Luke-workspace/OUTPUT
```

Figure 25: Argument file

Figure 25 shows the argument file for android, args_for_android.txt. There is one more similar file for iOS also, args_for_ios.txt. This file is passed as an argument for the starting script. Argument files usually contain command line options and test data path, one option or data source per line. They can contain any characters without escaping, but spaces in the beginning and end of lines are ignored. Additionally, empty lines and lines starting with a hash mark (#) are ignored:

- execTest.sh is the starting script. This is also stored under src folder. To start the test execution by a continuous integration system, it is necessary to have a start-up script. This script is also useful for manual execution of the script, especially with large number of command line options. The starting script is presented in Figure 26.



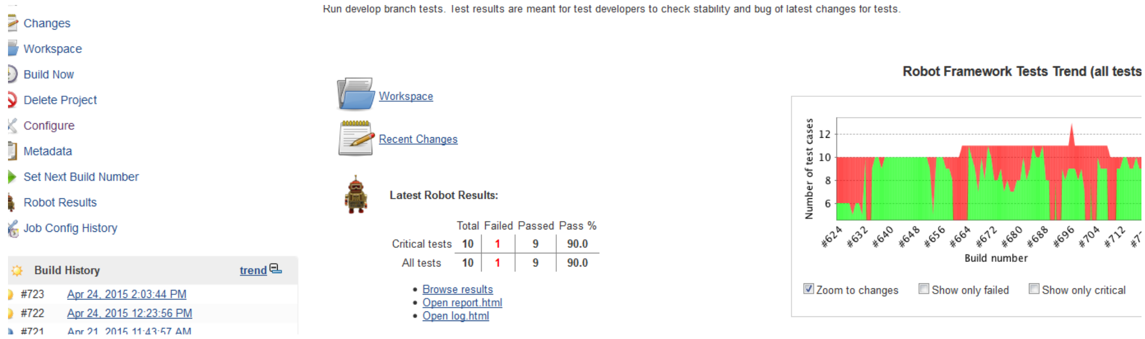
```
        ENV_IN_USE=DEV
        ENV_NAME="Development"
        ;;
    "staging")
        ENV_IN_USE=ST
        echo "using Staging Environment..."
        ENV_NAME="Staging"
        ;;
    "prod")
        ENV_IN_USE=UAT
        echo "using Production Environment..."
        ENV_NAME="Production"
        ;;
    "test")
        ENV_IN_USE=TEST
        echo "using Testung Environment..."
        ENV_NAME="Testing"
        ;;
    *)  echo "Unknown environment check!!!"
        exit 1
        ;;
esac


# Start Appium server:
pkill -9 node -f .*/appium
appium > $TA_OUTPUTDIR/appium_server.log 2>&1 &
sleep 5

pybot \
    --variable ENV_IN_USE:${ENV_IN_USE} \
    --variable ENV_NAME:${ENV_NAME} \
    --variable FOLLOW:TRUE \
    --variable FOLLOW_CHS:TRUE \
    -A ${argumentFile} \
    ${testCasesLocation}

# Kill Appium appium:
pkill -9 node -f .*/appium

echo ""
```

Figure 26: Starting script

Figure 26 presents the contents of the starting script. It accepts two arguments. The first one is the argument file that is explained above. The second argument is the application’s current environment. Depending on the environment specified the scripts are executed in the specified environment.

7.2 Test Case Execution

Test cases for some critical scenarios are implemented for this study. Either the test suite as a whole or just some test cases are selected for execution. Test execution is started by using the command pybot. This command is for executing the scripts in Python. Normal flow of execution is from the top-level test suite. Figure 27 presents the status of the current testing.



Run develop branch tests. Test results are meant for test developers to check stability and bug of latest changes for tests.

Build History trend

#723	Apr 24, 2015 2:03:44 PM
#722	Apr 24, 2015 12:23:56 PM
#721	Apr 21, 2015 11:43:57 AM

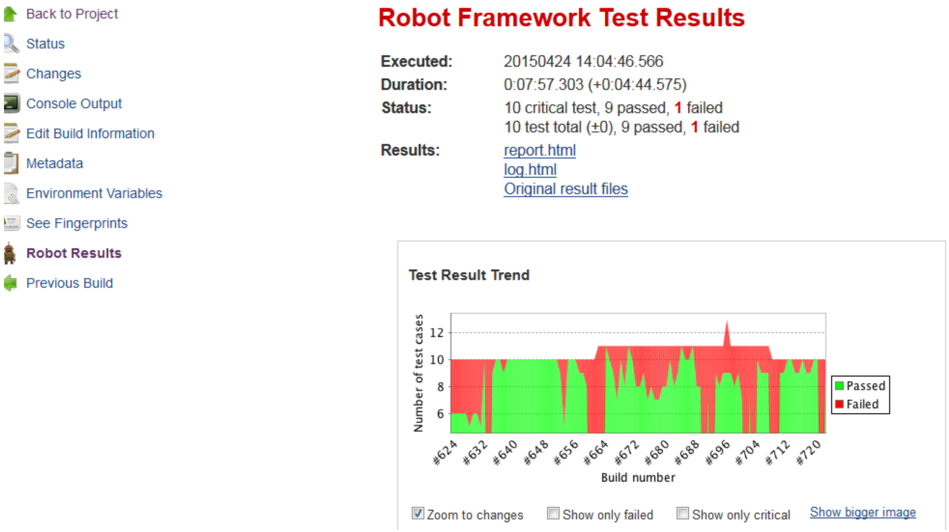
Latest Robot Results:

	Total	Failed	Passed	Pass %
Critical tests	10	1	9	90.0
All tests	10	1	9	90.0

- Browse results
- Open report.html
- Open log.html

Figure 27: Project Test Status

Figure 27 demonstrates the test status for the project. It shows how many test cases run and how many passed/failed. Test report is generated in various other formats. Robot Framework Plugin collects and publishes the test results of Robot Framework in Jenkins. The test results of robot framework is shown in Figure 28.



Robot Framework Test Results

Executed:	20150424 14:04:46.566
Duration:	0:07:57.303 (+0:04:44.575)
Status:	10 critical test, 9 passed, 1 failed 10 test total (±0), 9 passed, 1 failed
Results:	report.html log.html Original result files

Figure 28: Robot Test Results

Figure 28 is the robot framework test results. It shows the Build number vs. Number of test cases graph. Also it shows the report formats available for downloading the test results. Command line output is the most visible output. For detailed analysis of the test results separate output files are needed with the test execution information in the form of XML, higher level report of the execution in the form of HTML, and a detailed log file.

The test execution status shows that it is possible to automate the test cases with the selected tools and continuous integration environment.

8 Cost-Benefit Analysis

The cost and benefit of the automated testing undertaken in the company is calculated based on the test results. This calculated measure is used to evaluate whether the test automation has any benefit for the company. This section presents the technique used to measure the performance and also shows the computation results.

8.1 Computation Method

The software development process followed by the company is shown in Figure 12 in Section 3.2. From the figure it is evident that to develop software the company first works on the requirements and design part, then implementation part, after that testing, subsequently on acceptance testing and finally on production. Thus the total cost for developing the software (TC_{SD}) can be obtained by measuring the number of hours spent for requirements & design ($TC_{R\&D}$), implementation (TC_{IMPL}), testing (TC_{TEST}), acceptance testing (TC_{ACCP}).

Figure 12 also shows that for testing, acceptance and production, a number of cycles exist if bug fix is required. Thus testing effort cost ($TC_{TEST\ EFFORT}$) is calculated by the number of cycles of iterations of the testing period and acceptance testing period.

$$TC_{TEST\ EFFORT} = (numcycles_{TEST} * TC_{TEST}) + (numcycles_{ACCP} * TC_{ACCP})$$

Equation 1

Using the above equation, total cost for the software development is calculated as below:

$$TC_{SD} = TC_{R\&D} + TC_{IMPL} + TC_{TEST\ EFFORT} + (\text{average solved bugs} * AC_{BUGFIX})$$

Equation 2

In the above formula AC_{BUGFIX} is the average cost to fix bugs. However, for this research purpose only some critical cases are selected for automation and so the average cost to fix bugs is found negligible. Thus it is omitted for rest of the calculations and so the modified equation is:

$$TC_{SD} = TC_{R\&D} + TC_{IMPL} + TC_{TEST\ EFFORT}$$

Equation 3

The test set used for acceptance testing period is same as in testing phase. Hours spent for both phases are also same. Thus Equation 1 is modified as below:

$$TC_{TEST\ EFFORT} = (\text{numcycles}_{TEST} + \text{numcycles}_{ACCP}) * TC_{TEST}$$

Equation 4

From Equation 3 and 4, it is evident that the total cost for the software development is calculated by summing the total costs for requirements & design, implementation and testing. Total cost for testing comprises of the number of cycles for testing and acceptance testing and its associated costs.

8.2 Test Effort without Automation

The system has 120 use cases with 70% complex use cases. For each complex use case average of 20 test cases are required and each complex test case takes 10 minutes to test. For each simple use cases average of 10 test cases are required and each simple test case takes 5 minutes to test. TC_{TEST} is determined by calculating the hours spent for test execution.

Hours spend to test the entire system (TC_{TEST})

= (120 * 0.7 * 20 * 10) + (120 * 0.3 * 10 * 5)

= 16800 + 1800

= 18600 minutes

= 18600 / 60 = 310 hours

Average cycles for testing period is 20 and average cycles for acceptance testing is 5 as per experience. Thus total cost for testing effort $TC_{TEST\ EFFORT}$ is calculated by applying these values to the equation 4.

$$TC_{TEST\ EFFORT} = (20 + 5) * 310 = 7750 \text{ hours}$$

Right now the project has 6 resources and among them 4 are full time workers. The implementation period lasts for 6 weeks i.e. 240 hours. So, total cost for software development, without employing test automation, is determined using Equation 3.

$$\begin{aligned} TC_{SD\ WITHOUT\ TEST\ AUTOMATION} &= (TC_{R\&D} + TC_{IMPL}) + TC_{TEST\ EFFORT} \\ &= (4 * 240) + 7750 \\ &= 960 + 7750 \\ &= 8710 \text{ hours} \end{aligned}$$

This result shows that it is impossible to attain 100% test coverage with a release cycle of three months.

8.3 Test Effort with Automation

Time taken for manual testing and automation testing is measured by observation. The measurement is recorded for a single functionality and the readings are presented in Table 9:

Table 9: Manual and Automated test time consumption

Test cases	**Manual testing time (sec.)**	**Automated testing time (sec.)**
Test case 1	120	10
Test case 2	60	9
Test case 3	60	8
Test case 4	120	13
Test case 5	120	10
Test case 6	60	8
Test case 7	120	10
Test case 8	60	8
Test case 9	120	10
Test case 10	120	10
TOTAL	960	96

Thus the percentage of reduction for automation is calculated from Table 9 as:
%Reduce = (960 - 96) / 960 * 100 = 90%

Hours spent to test with automation is calculated as:
$TC_{\text{TEST WITH AUTOMATION}}$ = (100% - 90%) * 310 = 31 hours

Thus $TC_{\text{TEST EFFORT WITH AUTOMATION}}$ = (20 + 5) * 31 = 775 hours

However, with automation one more factor is there to consider. That is, test maintenance cost and is set to 57% of $TC_{\text{TEST WITHOUT AUTOMATION}}$. But this cost is usually lower than 57% because all use cases need not undergo modifications after each release. On an average 30% of the use cases get some modification. Thus
$TC_{\text{TEST MAINTENANCE}}$ = 30% * 57% * $TC_{\text{TEST WITHOUT AUTOMATION}}$ = 0.171 * 310 = 53 hours

So, total cost for software development, with test automation, is determined by adding the maintenance cost also to the Equation 3. That is,

$$\begin{aligned} TC_{\text{SD WITH TEST AUTOMATION}} &= (TC_{\text{R\&D}} + TC_{\text{IMPL}}) + TC_{\text{TEST EFFORT}} + TC_{\text{TEST MAINTENANCE}} \\ &= 960 + 775 + 53 \\ &= 1788 \text{ hours} \end{aligned}$$

The reduction of cost if automation is employed = 8710 – 1788
= 6922 hours
= 6922 / 8710 * 100
= **79%**

The analysis shows that test automation brings 79% cost reduction compared to the manual testing cost. Though test automation has some additional costs like script creation and maintenance, these costs could be covered by two or three releases. In this regard, the test automation is a beneficial solution to the company.

9 Proposed Solution and Recommendations

The research analysis showed that the company can make a huge benefit by incorporating test automation in their testing cycle. From the cost-benefit analysis it is evident that by executing the tests manually full test coverage will not be possible with the company which adversely affects the software quality. The project follows Agile processes and hence it follows an iterative & incremental approach. This makes the manual test execution a complicated inefficient process. In order to deliver a quality product in time it is wise to adopt automation testing.

The research carried out an in depth analysis of the popular tools available for test automation. By analyzing the tools features using Plotytsia's (2014) eleven steps and comparing the tools with each other one tool is selected for implementation. The selection was made also by matching the features with the company's specification. The selected tool was Appium for mobile test automation.

To get more benefit from test automation it is very important to integrate the automated tests with a continuous integration environment. This makes sure that with every code commit to a repository triggers a new build by the continuous integration server. Then the server runs the automated tests. After the execution developers are notified of any bugs and test reports are generated. The research has already showed the continuous integration pipeline in Figure 19. The company can use this solution for implementing the continuous integration environment. The tools used are GitHub as code/artifact repository, Jenkins as continuous integration server and Robot Framework as test automation framework.

In short, this research recommends test automation for efficient testing and for delivering quality software. Also by using a key-word driven framework such as Robot Framework enables even non-technical resources to maintain the test scripts. This will come beneficial for the company in long run. Robot Framework also allows creating the keywords even before the UI implementation is done. Another recommendation is to have the continuous integration environment for getting the advantage of automation. Using Jenkins as continuous integration server will be beneficial for long time because Jenkins support numerous plugins. Therefore if company needs to change tools it can be done without changing the continuous integration environment.

10 Conclusion and Summary

This section concludes the research by presenting the main results obtained in the study. This section also details how the objective is attained by answering the three research questions presented in Section 1.1.

10.1 Summary

The research was conducted for Natural Resources Institute Finland (Luke Oy.). The company finds it hard to meet the software quality because of the short release cycle and inefficient testing processes. Company's applications are of traditional web applications and mobile applications. However, this study was done for the mobile applications because that is the area where company needs to ensure the efficiency the most.

The objective for the study is given in Section 1.1 as "How the company can effectively implement their quality assurance effort for mobile web applications?"

In order to attain this objective, the research answered the three research questions. The questions are listed below along with the answers.

1) How an efficient tool and a test framework could be chosen for the automation? Some of the most popular mobile automation tools are selected for comparison. Among them five tools that can test Android and iOS are chosen for detailed analysis. The study used Plotytsia's (2014) eleven steps to analyze the five tools. This analysis is presented in Section 5.3. Then they are compared with each other and selected the most matching one with the company's requirement that is presented in Section 5.2. The tool comparison is showed in Table 8 in Section 5.4. The automation tool used in this study is Appium as it is best matched with the company's specification.

   Test framework selection is presented in Section 6.1. The conditions for selecting the right test framework depend upon the project requirements. Thus six conditions are specified for the test framework to satisfy based on the project requirements. Robot Framework, a keyword-driven testing framework is selected as the testing framework.

2) How the chosen tool and framework could be incorporated in the continuous integration?
   For getting full benefit from the test automation it is essential to integrate a continuous integration environment in the software development cycle. In this regard, the research used a continuous integration environment. For the continuous integration server this study selected Jenkins because of its popularity, vast userbase, open-source, numerous plugins available etc. The selection procedure for Jenkins is presented in Section 6.2. The testing tool selected is Appium and the selection procedure is detailed in Section 5. As an artifact repository this study chose GitHub because of its popularity, features, open-source etc. The selection of GitHub is showed in Section 6.3. The continuous integration pipeline used for the company is showed in Figure 19 under Section 6.4.

3) How the chosen tool and framework could be implemented in end-to-end testing?
   End-to-end testing is a methodology to verify that the application flow is as expected from start to finish. It ensures that the integrated components are functioning properly.

   With the integration of continuous integration pipeline in the development cycle the full processes after code commit is automated. That is when the code repository identifies any code changes the continuous build system is triggered for a new build. Then the automated tests are executed on the new system and test reports are generated. If the newly introduced changes have any bugs associated with it then that is notified to the developer. Also the test run status is also sent to the stakeholders. Thus the full development cycle is implemented in the real world and ensured that all the components are working as per the expectations.

Coming to the main objective of the study, the goal to conduct a proof-of-study to improve the company’s quality assurance measures is successfully accomplished by implementing mobile test automation for the testing process. Additionally end-to-end testing is realized in the real environment. Finally, a cost-benefit evaluation is conducted to analyse if the improvement measures is profitable for the company. The analysis showed that automation testing cost is reduced by 79% compared to the manual testing cost. Though test automation needs more time for test script creation and maintenance which is a bit costly, the company can make up this cost in two or three release cycles.

10.2 Conclusion

Quality of software is ensured by conducting a thorough testing. In Agile methodologies the release cycle is short and each release accompanies many changes. In this regard it is essential to perform integration testing. If the testing time is short then it is not possible to test all scenarios by using manual testing. This can adversely affect the software quality. Test automation is a time saver in these situations.

Considering the advantages of automation testing, it is implemented in real environment. The literature review recapped basics of testing, important concepts, tools, frameworks etc. From the literature study key-word driven testing is selected as the testing approach. Appium is selected as the testing tool after analyzing its features and comparison with other popular tools. Robot Framework is selected as the testing framework because it follows key-word driven testing approach. Jenkins is opted as the continuous integration server considering its popularity and other features. GitHub is used as the code / artifact repository.

The results of the test execution are used to compute the test automation cost and the cost reduction by using test automation. These computations are presented in Section 9. The computations show that the total cost of software development is reduced by 79% by introducing test automation in place of manual testing. It is also realized that test automation brings some additional cost in terms of test script creation and its maintenance. However, these costs could be covered in two or three release cycles and company could start harvesting the benefits from next cycle onwards. This also brings additional advantage over manual testing that the test scripts serves as a future reference material for future testing. Thus it can be concluded that test automation brings cost reduction to the software development process and in long run it brings return on investment to the company.

10.3 Further Research

The current research was conducted for automating mobile applications. However, automating web applications is another task that could be performed in future. While conducting the current study, the tools and framework selection is performed by keeping this possibility also in mind. Hence without changing any of the current implementation, company could add web application automation also to the testing environment. For instance, Robot Framework has a Selenium Plugin for web browser testing.

References

Literature

Bentley, J.E. 2004. Software Testing Fundamentals - Concepts, Roles, and Terminology, Wachovia Bank, Charlotte NC.

Bradbury-Huang, H. 2010. “What is good Action Research? Why the resurgent interest?” in Action Research 8(1).

Burnstein, I. 2003. Practical software testing. New York: Springer-Verlag.

Carmi, A. 2014. Automated Visual Software Testing: The Missing Link in Software Test Automation.

Davison, R.M., Martinsons, M.G., & Kock, N. 2004. Principles of Canonical Action Research, Information Systems Journal 14(1), pp. 65–86.

Easterbrook, S.M., Singer, J., Storey, M. & Damian, D. 2007. Selecting empirical methods for software engineering research. In: Shull, F. & Singer, J., (Eds.). Guide to Advanced Empirical Software Engineering. Springer, 2007, 285–311.

Elden, M., & Chisholm, R. 1993. Emerging varieties of action research: Introduction to the special issue. Human Relations, 46(2), 121–142.

Faught, D.R. 2004. Article: Keyword-Driven Testing. Sticky Minds. Software Quality Engineering.

Fewster, M. & Graham, D. 1999. Software Test Automation: Effective use of test execution tools. Great Britain: Addison-Wesley.

Florac, W.A. 1992. Software quality measurement a framework for counting problems and defects. Technical Report CMU/SEI-92-TR-22, 1992.

Gelperin, D. & Hetzel, B. 1988. "The Growth of Software Testing". Communications of the Association for Computing Machinery 31 (6).

Gittens, M., Lutfiyya, H., Bauer, M., Godwin, D., Kim, Y.W. & Gupta, P. 2002. An empirical evaluation of system and regression testing. In CASCON ’02: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, page 3. IBM Press.

Grindal, M. & Lindström, B. 2002. Challenges in Testing Real-Time Systems. In Proceedings of the 10th International Conference on Software Testing Analysis and Review (EuroSTAR’02).

Henry, P. 2008. The Testing Network: An Integral Approach to Test Activities in Large Software Projects. Springer, Heidelberg.

Holopainen, J. 2004. Regressiotestauksen tehostaminen, Pro Gradu, Tietojenkäsittelytieteen laitos, Kuopion Yliopisto.

IEEE Standard 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology. IEEE (The Institute of Electrical and Electronics Engineers) Standards Association, 1990.

IEEE Standard 729-1983. IEEE Standard Glossary of Software Engineering Terminology. IEEE, 1983.

IEEE Standard 982.1-1988. IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE, 1988.

Itkonen, J., Mäntylä, M.V. & Lassenius, C. 2009. How do testers do it? An exploratory study on manual testing practices, in Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 494–497.

Jenkins, M.A. 1985. Research Methodologies and MIS Research. In: Research Methods in Information Systems (IFIP 8.2 Proceedings), Mumford, E.; Hirschheim, R.; Fitzgerald, G. & Wood-Harper, T. (Eds.), Elsevier Science Publishers B.V., North-Holland, Amsterdam, 103 – 117.

Kaner, C., Bach, J., & Pettichord, B. 2001. Lessons Learned in Software Testing. John Wiley & Sons, Inc., New York, NY, USA.

Kaner, C., Falk, J. & Hguyen, H. 1999. Testing Computer Software, 2 nd edition, John Wiley & Sons, New York, pp. 124.

Kelly, M. 2003. Article: Choosing a test automation framework.

Kemmis, S., & McTaggart, R. 2000. Participatory action research. In N. Denzin & Y. Lincoln (Eds.), Handbook of qualitative research (2nd ed., pp. 567–605). Thousand Oaks, CA: Sage.

Lewin, K. 1946. Action research and minority problems. Journal of Social Issues (2) 34–46.

Miller, R. & Collins, C. 2001. Acceptance Testing, RoleModel Software, Inc.

Myers, 1979. G Myers. The Art of Software Testing. John Wiley & Sonc, Inc., New York.

Royce, W. 1970. Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26, pp. 1-9.

Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum, Upper Saddle River. NJ, Prentice-Hall.

Sommerville, I. 1996. Software Engineering, 5th Edition, Addison-Wesley, 445-462.

Sommerville, I. 2001. Software Engineering, 6th Edition, Addison-Wesley, 442-444.

Utting, M. & Legeard, B. 2006. Practical Model-Based Testing: A Tools Approach, Morgan-Kaufmann.

Wright, J. 2010. Hybrid Keyword Data Driven Automation Frameworks, ANZTB Conference.

Electronic references

Appium 2015. Referenced on 15 April 2015.
http://appium.io/

Bowes, J. 2012. Jenkins Continuous Build System. Referenced on 23 April 2015.
http://www.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/bowesjesse.pdf

Calabash 2015. Referenced on 15 April 2015.
http://developer.xamarin.com/guides/testcloud/calabash/introduction-to-calabash/

Calabash wiki 2015. Referenced on 30 April 2015.
https://github.com/calabash/calabash-android/wiki/06-webview-support

CloudMonkey 2015. Referenced on 15 April 2015.
https://www.cloudmonkeymobile.com/monkeytalk

Conformiq 2014. Automated Test Design – Model-Based Testing. Referenced on 21 April 2015.
http://www.conformiq.com/model-based-testing/

Dick, B. 2002. Action research: Action and research. Referenced on 19 March 2015.
http://www.aral.com.au/resources/aandr.html

Extreme software testing 2009. Software testing history. Referenced on 21 March 2015.
http://extremesoftwaretesting.com/Info/SoftwareTestingHistory.html

Fowler, M. 2006. Continuous Integration. Referenced on 23 April 2015.
http://www.martinfowler.com/articles/continuousIntegration.html

Frank 2015. Referenced on 15 April 2015.
http://www.testingwithfrank.com/

Kaner, C. 2006. "Exploratory Testing" (PDF). Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL. Referenced on April 21, 2015.
http://www.kaner.com/pdfs/ETatQAI.pdf

KIF 2015. Referenced on 15 April 2015.
https://github.com/kif-framework/KIF

Leveson, N. & Turner, C.S. 1993. An Investigation of the Therac-25 Accidents. IEEE, 26(7), 18-41. Referenced on 15 February 2015.
http://courses.cs.vt.edu/cs3604/lib/Therac_25/Therac_1.html

Plotytsia, S. 2014. Article: How to Choose the Right Mobile Test Automation Tool? Referenced on 19 March 2015.
http://www.testlab4apps.com/how-to-choose-the-right-mobile-test-automation-tool/

RobotFramework 2015. Robot Framework Introduction. Referenced on 23 April 2015.
http://robotframework.org/

Robotium 2015. Referenced on 15 April 2015.
https://code.google.com/p/robotium/

Selendroid 2015. Referenced on 15 April 2015.
http://selendroid.io/

Sikuli 2015. Referenced on 15 April 2015.
http://www.sikuli.org/

SOASTA 2015. Referenced on 05 May 2015.
http://www.soasta.com/products/touchtest/

Software Testing Mentor 2015. Referenced on 05 May 2015.
http://www.softwaretestingmentor.com/automation/manual-vs-automation/

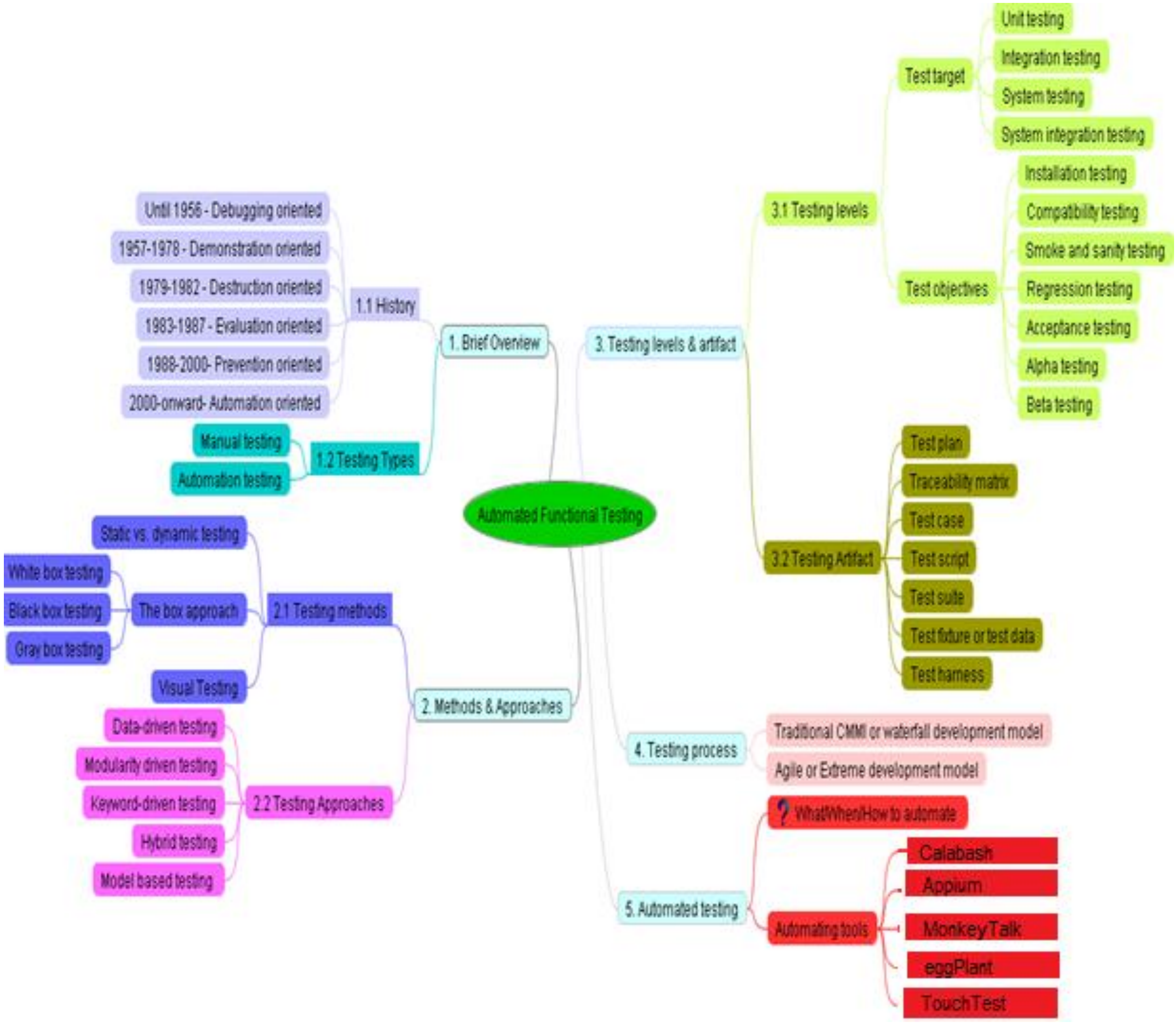
TestPlant 2015. Referenced on 05 May 2015.
http://www.testplant.com/eggplant/

The UK Passport Agency report 1999. The passport delays of Summer 1999 - Comptroller and Auditor General, National Audit Office. Referenced on 15 February 2015.
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=rja&ved=0CEcQFjAD&url=http%3A%2F%2Fwww.nao.org.uk%2Fidoc.ashx%3FdocId%3Df999cace-5d1b-4e57-81dd-e3d22b8b6687%26version%3D-1&ei=kQDRUM-RBMTE4gSBnoGYAQ&usg=AFQjCNG7lTMAD_TulgAehtMKqoyiITPRnw&sig2=o_UzrrSFQJ8VEgu1LIZBeQ&bvm=bv.1355325884,bs.1,d.bGE

Velocity Culture 2011. Jon Jenkins’s talk “Velocity Culture”. Referenced on 1 February 2015.
http://www.slideshare.net/oreillymedia/velocity-2011-jon-jenkins-velocity-culture

Appendix 1. Mind map

Mind map for the topic Automated Functional Testing. This serves in the literature study of the thesis.



Appendix 2. Questions for SWOT Analysis

Questions asked for creating the SWOT analysis are listed below

STRENGTHS
What do you do well with the software product?
Any advantages over the competition, if any?
What are the internal assets of the company?
Any other positive factors that add value to your business?
WEAKNESSES
What factors limit your ability to deliver quality product?
What are the improvement areas to attain the objectives?
Is the testing conducted efficiently?
What are the skill set of your testing resources?
What does your business lack with respect to technology, expertise, skill set etc.?
OPPORTUNITIES
Any recent market changes or market growth that gives you an opportunity to flourish?
Is that opportunity ongoing or is there any time factor?
Is your business have a positive feedback?
Is there any other opportunity exist in the market that benefit you?
THREATS
What factors put your business at risk?
Is there any competition that adversely affects you?
Any new product that make your service obsolete?
Any shift in target customer’s behaviour that reduces your sales?