Final thesis

GUI test automation for Qt application

by

Lei Wang

LIU-IDA/LITH-EX-A--15/053--SE

2015-09-15
Final Thesis

GUI test automation for Qt application

by

Lei Wang

LIU-IDA/LITH-EX-A--15/053--SE

2015-09-15

Supervisor: Nima Aghae Ghaleshahi
           IDA, Linköping University

Examiner: Ahmed Rezine
          IDA, Linköping University
Abstract

GUI test automation is a popular and interesting subject in the testing industry. Many companies plan to start test automation projects in order to implement efficient, less expensive software testing. However, there are challenges for the testing team who lack experience performing GUI tests automation. Many GUI test automation projects have ended in failure due to mistakes made during the early stages of the project.

The major work of this thesis is to find a solution to the challenges of establishing new test automation projects and determine an appropriate method to build GUI tests automation. Due to the particularity of GUI tests automation, keyword driven test approach is introduced in this thesis. The advantages and disadvantages are shown by undertaking comparison with the Record and replay testing method and the data driven testing method. The research also includes a feasibility study of GUI tests automation. The analysis report points out which kinds of test projects should be automated and which kinds should not be automated.

Test automation tool selection is a critical step for an test automation project. This thesis shows the correct procedure for selecting a testing tool and explains the strategies of testing tool selection. It also summaries the mistakes made during the tool selection process. Several classical types of testing tools which support Qt software testing are compared and evaluated. Based on the in-depth analysis and comparison of testing tools, the thesis explains how the different tools fit with the different test projects.

The implementation procedure of a test automation is demonstrated in this thesis. The procedure includes test case design and testing framework implementation. The test script is created based on the structure of a keyword driven test framework. The thesis also discusses several interesting topics related to GUI tests automation for future research.
ACKNOWLEDGEMENTS

I got help from many people to complete this thesis. First and foremost I would like to express my sincere thanks to Professor Ahmed Rezine whose supervision and support make me overcome all the difficulties of this project. Without his patience and guidance, the thesis would not be finished.

Thanks Movial Oy for giving me the study opportunity of test automation. It was my first time to work for a subject in real testing industry. The work experience in the company helped me to understand more about this question. I own my deepest gratitude to my project manager Andrew Sheremetyev. He inspired me to realize the importance of test automation in software manifesting industry.

I also want to thank Squish and Ranorex which gave me the permission to try their product for a month. I leant a lot about the testing tool from their high quality software. Without their help, I wouldn’t have such deep understanding of the testing tool.

I am deeply indebted to all the people who provide guidance and explanations to my questions of this project. Thanks for all the support.
Table of Contents

Chapter 1   Introduction.......................................................................................................................... 1
  1.1.  Background...................................................................................................................................... 1
  1.2.  Problem description.......................................................................................................................... 1
  1.3.  Method............................................................................................................................................. 2
  1.4.  Thesis structure .............................................................................................................................. 3

Chapter 2  Software testing introduction................................................................................................. 4
  2.1.  Software development model .......................................................................................................... 4
  2.2.  System testing................................................................................................................................. 5
  2.3.  Regression testing ............................................................................................................................ 6
  2.4.  Black box testing ............................................................................................................................ 6
  2.5.  Definitions ...................................................................................................................................... 6

Chapter 3   GUI testing automation .......................................................................................................... 7
  3.1.  The particularity of GUI testing ...................................................................................................... 7
  3.2.  GUI test automation method .......................................................................................................... 8
    3.2.1.  Record and replay ...................................................................................................................... 8
    3.2.2.  Problems of Record and replay .............................................................................................. 9
    3.2.3.  Use of Record and replay ........................................................................................................ 9
    3.2.4.  Data driven testing and keyword driven testing ..................................................................... 10
    3.2.5.  Advantages of keyword driven test ........................................................................................ 11
    3.2.6.  Disadvantages of keyword driven test .................................................................................. 12

Chapter 4   Considerations about test automation ................................................................................... 13
  4.1.  Advantages of test automation ....................................................................................................... 13
  4.2.  Disadvantages of automated tests .................................................................................................. 14
  4.3.  What kinds of tests are suitable for automation? ............................................................................ 15
  4.4.  What kinds of tests are not suitable for automation? ..................................................................... 16
6.4.2. Handling of incidents ................................................................. 46
6.4.3. Time setting .............................................................................. 46
6.4.4. Pre- and post-processing ......................................................... 47
6.4.5. Report of testing result .............................................................. 47
6.5. Discussion ................................................................................... 49
6.5.1. Analysis of testing report .......................................................... 49
6.5.2. Implementation difference ....................................................... 49
Chapter 7 Conclusion and future work .................................................. 52
References ....................................................................................... 54
List of Figure

Figure 1 V-model .................................................................................................................. 5
Figure 2 Record and replay procedure .................................................................................. 8
Figure 3 Keyword-driven test ............................................................................................... 11
Figure 4 Overview of the tool selection process ................................................................. 19
Figure 5 Testing tool Architecture ...................................................................................... 26
Figure 6 UI inspector ............................................................................................................. 28
Figure 7 UI object identification ............................................................................................ 29
Figure 8 Login window ......................................................................................................... 38
Figure 9 Student Info window ............................................................................................... 38
Figure 10 Product specification generation process ............................................................ 39
Figure 11 Testing framework architecture ............................................................................ 42
Figure 12 Testing report ........................................................................................................ 48
Figure 13 Table converter ..................................................................................................... 51
List of Table

**Table 1** Script languages which are supported by the testing tools .............................................. 22

**Table 2** Document information of manual test case........................................................................ 40

**Table 3** Document information of test automation case ................................................................. 41

**Table 4** Data table for TC2 ......................................................................................................... 50
Chapter 1

Introduction

Graphical user interfaces (GUI) are widely used in software applications. They make the applications easier to use and user friendly. In modern software applications, around 50% of the application code is for GUI implementation [1]. The quality of the GUI is a critical factor for the security, stability and reusability of software. For the individual customer, an ingenious user interface will have a good effect on the user experience. Thus, GUI testing has become necessary and important for the quality of a software application.

1.1. Background

The application needs to have various kinds of input methods and to be easy to use. This requests complex graphical user interface design, which again means huge amount of testing. As labor costs are expensive and the demanding of high quality of software are increasing, a GUI test automation would be a good investment for any software manufacturing company. There is a growing trend in software industry that a lot of companies attempt to automate the tests of their products so the workload of the tester will be reduced.

Many software manufacturing companies have a long history of using Qt framework for software development. They may also use other software development frameworks, but this thesis only focuses on the test automation for the Qt application.

1.2. Problem description

The company which provides technology services with software development usually produces a wide variety of applications. There may be applications running on Linux, iOS, Android and Windows. The software application is usually updated to a new version every now and then. When it is updated, the tester needs to go through the entire test of this application. It can take several days for a tester to finish several hundreds of test cases. The initial purpose is to automate the manual test, so the company can save the labor cost for the testing. It also makes the tester’s work more valuable.

Many software manufacturing companies have never designed or automated any GUI test before. User interface testing of all the applications is normally done manually. Hence, it is necessary to research how to start to automate tests. There are many issues that need to be studied and
appropriately solved at the beginning of the test automation project. For example, what kind of software is suitable to be tested automatically? Is it worth it to automate these manual test cases? How to choose the appropriate testing tools? Any mistake made which is related to these issues may cause test automation to fail.

The purpose of the test automation project is not only to automate the manual test, but also to gain test automation experience which will benefit all future GUI tests. So the thesis is not only about the procedure of test automation, but also includes the lessons learnt from this experience. All preparation work for test automation project will benefit from that experience.

1.3. Method

To achieve the goal of this thesis and identify the solution, the following approaches will be used:

1. Literature study. Books which systematically expound on the theory of testing automation will be studied to gain tips and advice for test automation building. The reading list also includes reports from testers and test managers who are in charge of test automation of real business projects. These reports describe how their GUI test automation projects are developed and maintained. They also describe many common problems that the testers encountered during the test automation implementation.

2. Research in testing tool. To select the proper testing tool, it is necessary to have a deep understanding of testing tool architecture. Research papers for specific testing tools will illustrate the general structure of a GUI test tool. Summarizing the content of these papers will help us learn the concept of the testing tool modules and their usage.

3. Testing tool comparison. Some articles mention the methods of tool selection. User guides for testing tools also highlight the features of their products. All these documents will help us to set up the criteria for tool comparison. All the information related to the testing tools will be used for tool comparison. The information includes the content of the user guide, feedback of the tool’s usage from other testers, and discussions about the GUI testing tool in software testing forum and software testing conference.

4. Practice using testing tools. The purpose of the practice is to become familiar with the tools’ usage in test automation projects and to compare the testing tools in detail, so that a proper decision on tool selection can be made eventually.

5. Case study. Select a software application as an example to show the implementation procedure of a test automation project. The implementation also demonstrates the testing methodology and tool usage which is adopted in this thesis.
1.4. Thesis structure

Chapter one is the background of this project. It includes the problem description, the goal of the thesis and the study approach.

Chapter two introduces the basic knowledge of test theory which is relevant to this project. The software development model, which is introduced in this chapter, shows the different kinds of testing activates in different developing stages. Among them, system testing and function testing will be performed in the test automation implementation in chapter six. Besides these testing categories, GUI testing automation also belongs to black box testing, and regression testing in most cases. Chapter two introduces the basic concepts of these testing categories.

Based on the basic test theory of chapter two, chapter three points out the particulars and challenges of GUI test automation. It also introduces three testing approaches which are commonly used in GUI test automation. Among them, keyword driven testing is the approach which will be adopted for test automation implementation in chapter six.

Chapter four features a discussion about the advantages and disadvantages of test automation. It reaches a conclusion on what kind of test is suitable for automation and what kind of test is not suitable. The study and discussion in chapter four and chapter five is for answering the questions related to the preparation of test automation project. The preparation questions include whether the manual test should be automated, and what testing tool we can use.

Chapter five expounds the whole selection procedure of proper testing tools. Firstly, it describes the pitfalls that the tester may face during the tool selection. Then, different kinds of testing tools are compared from various aspects. At the end of the chapter, the decision on which tool is the correct choice is made.

After the study of chapter five, a test tool is selected and the test automation implementation is ready to start. Chapter six shows the basic GUI test automation procedure by implementing several test cases of an example application. A keyword driven test based testing framework is built to generate test cases. The bottom modules of the testing framework are created by the testing tool. How to build these bottom modules is explained in chapter five, and chapter six mainly focus on how to create the testing scripts based on these modules.

Chapter seven is the conclusion of test automation project. It also discusses possible improvements.
Chapter 2

Software testing introduction

In a perfect world, every function works exactly as the customer requires when the software is developed. But the reality is different. From the abstract customer requirement to the acceptable product, software development is a long and complex procedure including many different steps. Each step may cause defects. To identify these errors and guarantee the software quality, software testing is an important part of the software development process.

2.1. Software development model

With the technology developing, software structure is becoming more complex and the size of application code is increasing. In order to manage software development activities in an effective way, the contemporary software development process is usually implemented in accordance with a certain software development model. The model is used as a framework which illustrates a particular sequence of development activities and the corresponding testing events.

There are various kinds of software development models. Some popular ones are for example the V-model, the Rapid Application Development (RAD) model, the Test Driven Development (TDD) model, and the Agile Software Development model. They all have their own advantages and disadvantages. For our project, the V-model is used as the software development model. The V-model is shown in Figure 1
Figure 1 V-model [2]

The left part of the figure shows the software development steps. The right part shows the corresponding testing activities. They represent the classifications of tests which are used to detect the defects of certain levels. These testing activities may not happen in a fixed order. Some of them may not even be used at all in an application development process.

The testing activities are divided into four phases. As Figure 1 shows, the unit testing is used to verify whether a small fragment of source code works or not. This part of code is independent from others and can be tested individually [3]. The integration testing tests the interface of an individual module and checks whether different modules can work together in proper way [3]. System testing tests whole entirety of software application. This thesis is mainly about system testing. The details of system testing will be discussed in chapter 2.2. Acceptance testing is used as a presentation for customers. So customers know that the application meets their requirements [3].

2.2. System testing

System testing is used to test the product as a whole system in the practical execution environment. It includes the hardware platform, supporting software and input data etc. System testing is the general term for multiple sublevel tests [4]. It includes security testing, performance testing and stress testing etc.

Functional testing is one type of system testing. It belongs to the black box test. During the testing process, the tester does not consider the internal structure of the software, but only checks whether the application can process the input data appropriately and generate the correct output information [3]. It verifies the application from the user point of view [4].

Most test cases in this thesis are about the operations of graphical user interfaces. They all belong to functional testing.
2.3. Regression testing

Regression testing does not belong to any of the testing phases previously discussed. It is used when the software defect is fixed or the original functionalities have been adjusted. Especially when there are new functionalities added, or the application is updated to the newer version, regression testing is performed to ensure that the new changes will not bring any error to these original functionalities which previously passed the tests [5]. Although regression test will not be implemented in the study case of the thesis, it is a very important for GUI test.

2.4. Black box testing

The name “black box” is the description of the testing methodology. The test process is like manipulating a black box. The tester does not need to know what the inside of the box looks like, for example the structure of the application code. The tester only needs to provide the user’s input and verify whether the output meets the expectation [4]. The tester does not need to understand how the output data are created.

2.5. Definitions

The definitions which will be used in this thesis are introduced in this chapter.

Application under test (AUT) is the application or system which is tested in this test automation project [6].

Test case (TC) is a segment of a test. It is usually used to test one feature of AUT. It includes the complete testing operations to test this feature and the verification operation to decide whether it is passed or failed [6].

Test Suite (TS) is a set of test cases which have a similar testing target. It may be used as the collection of the whole test cases of AUT, or the collection of test cases that belong to a certain category [6].
Chapter 3

GUI testing automation

3.1. The particularity of GUI testing

As the front end of an application, the GUI provides the user with a visual interface to interact with the system and to reduce the complexity of the interaction. However, this is also very challenging for the software test, making GUI test quite different from other types of tests.

Some specificities of GUI testing are listed below.

1. Both the input and the output of GUI testing are complex graphical control components of software interface. Graphical control components are commonly called UI objects. Each UI object contains a set of properties. Every property has its own value. UI Objects have a wide range of types. Different types of objects have different kinds of properties. Converting a wide variety of information to the same data format so that the computer can handle the input and output data in the batch is a challenge.

2. Each UI object has its own state machine [7]. The state of an object may be changed in a specific time. For example, the state of a window can be available, appear but not be available, or not appear. Unlike manual tests which the human can monitor the test execution by having a quick glance at the screen, the wide variety of objects increases the difficulty of how the computer can monitor the test processing situation in real time.

3. The interface objects are usually organized in tree structures [7]. For example, the main window of a notepad includes the objects “file” menu, “edit” menu and so on. Each menu also contains different options. For example, “file” menu includes a “save” option and a “close” option etc. The “save” option is only available after the user clicks the “file” menu. The dependence between different objects needs to be considered in the test implementation.

4. As discussed in the second chapter, most the automation of tests are built for regression testing. GUI regression tests are also different from the traditional regression tests. It is very
common that the UI objects are changed somewhat in a new software version. For example, the title, size or color of a button may change slightly. The result is that the test program no longer recognizes the UI object. This may result in not executing parts of the testing script or failing many test cases. How to build robust and maintainable GUI automatic tests is the biggest challenge in this project.

3.2. GUI test automation method

The testing tools are used to reduce the complexity of test automation implementation. The various tool features can not only decrease the script work but also provide the test automation approaches.

We focus on three main GUI test automation methods. Most GUI testing tools have corresponding features which can be used to support these methods for the testing project.

The test methods introduced in this thesis are Record and replay, Data driven test and Keyword driven test.

3.2.1. Record and replay

The Record and replay method is commonly used at the beginning of a GUI test automation, especially when the tester does not have much experience with GUI testing automation. If the software product was tested manually before and the company wants to automate the existing test case, the quick and easy way to automate the test case is to use the Record and replay method. But due to the significant defects of this method, it is not considered as test automation. However, it is still very useful for most GUI test automation projects.

The basic idea of the Record and replay method is to use the testing tool to capture all the tester’s actions (for example keyboard typing, mouse clicking etc.) during a manual test. The testing tool translates the recorded actions to test script and can run the script later as play-back. In this way, the manual test is kind of automated by the recorded script.

Figure 2 shows the Record and replay procedure.

![Figure 2 Record and replay procedure](image)
3.2.2. Problems of Record and replay

1. Poor test performance

During the Record and replay process, every human action is recorded. It requires high accuracy for each step. If there is a wrong test step during the recording, the tester may need to stop the recording and start it again from the beginning.

The tester may make some unnecessary actions during the recording procedure [8]. Therefore, the generated script includes redundant mouse clicking and keyboard typing actions. The script may also records the long and unnecessary waiting period between each step.

2. Script is hard to read and maintain [8]

A test script is automatically generated by the test tool. Before the script is optimized, it has no structure and no comments. It may include coordinates, long unreadable object names and a long list of similar operations.

This kind of script makes the maintenance work very complex. The tester has no idea what the recorded script tests in a short period, especially for these people who just take over the project from other people. If the script needs to be updated, it is also hard to figure out which part to change and which part not to change.

3. Cannot make completed test case

The generated scripts do not usually include steps like test verification, log generation or different error message handling. It means the tester must make some necessary changes to the recording scripts according to the test requirements. Some test tool features allow the tester to pause the recording and add the script during the recording procedure, or provide additional functions to make the modification easier. However, simply test recording would not create a completed test case.

4. Poor reusability for similar tests

The test script is created by the specific manual test. Therefore, it cannot be reused for other test case. If the tester wants to get the script of other test case, he has to perform the recording actions again. And since the generated script has no code structure, it is also hard to reuse even part of test script for other test case. When the amount of test case increases, the amount of test scripts becomes huge.

3.2.3. Use of Record and replay

1. Quick start of the test automation project

Although recorded script cannot be used as a completed test case and has many drawbacks, the Record and replay method is still a popular test method [8]. For a company that is just starting a new test automation, especially a company without much test automation experience, Record and replay is an easy approach to quickly start a project. This approach has similar way of thinking to
a manual test and very low skill requirements of the tester. Although the recorded script needs to be modified later, this test approach can still enhance the tester’s confidence of the implementation of the test automation project and quickly familiarizes the tester with the testing tool used in the project.

2. Preparation of formal test

Due to the simple duty of the Record and replay approach, most test automation tools have a feature to support this method. Since the testing operation of UI objects is translated into the recorded scripts, this method can be used as a preparation for test automation. By using this method, the tester knows the name of each UI object on the user interface and gets some test script to start the test script creation. For some testing tools, the tester first has to record to get object IDs and generate the object map files. The object map feature will be discussed in chapter 5.5.5.

3.2.4. Data driven testing and keyword driven testing

Many GUI test automation projects fail not only because of the complexity of the implementation procedure, but also because of the high maintenance cost [6]. Therefore, to build maintainable GUI tests is important for the success of a project. Obviously, the modification of recorded script is not enough. The test scripts are like software programs. The script should have structure and be reusable for many test cases.

Based on this idea, the data driven test method is mentioned in many GUI test automation projects [9] [10] [11]. The test data are separated from the test script. The implementation includes the files which contain the test data and the script which implements the logic of the test case. When the test case is executed, the test script reads the data from an external data source, and then performs the test case based on the test data [9]. The testing data does not only include the input data, but also contains the data which is used as expected value and compared with testing output. The testing actions may vary with different input data. In this way, the same test script can be used to run a number of different test cases. For the data driven test method, the script file and data file are updated separately. A testing script file may bind with multiple groups of testing data and generate various test cases. Therefore, the utilization of test scripts is improved. Since the testing script is no longer unstructured code, testing script maintenance becomes easier [10].

Here is an example of a data driven test. The test case is used to verify the login function of a chat application. With a different user name and password which are used as test data, the same test script can be used to generate many test cases.

Although testing scripts and testing data can be maintained by different people in the data-driven test method, the person who is in charge of testing data maintenance still needs to have a good understanding of testing script. To improve the flexibility and reusability of testing script, another testing approach was created as the enhancement of data driven testing. The testing method is called keyword driven test. It is used based on the idea of the data driven test. The keyword driven test also makes script maintenance easier.

The keyword driven test does not only separate out the testing data from the script, it also separates out the data handling logic by encapsulating the testing operation into keywords. For each keyword, testing step descriptions and specific implementation details are separated. The testing step implementation is encapsulated into the keyword, and the keyword name explains what function the
keyword performs. Therefore when the tester creates the test case, he can just use the various keywords from the keyword database and does not need to think about the implementation details of the test steps.

For example, in the previous login example, the login function can be implemented as a keyword. It can be used with the keyword “user name verification” to generate one test case. It can also be used with the keyword “user registration” in another test case. In this way, when the UI changes, the tester only needs to modify test scripts but there is no need to modify the keywords. The following figure shows the idea of the keyword-driven test.

![Figure 3 Keyword-driven test](image)

The test script is built up by various keywords. One keyword may be contained in different test scripts. The same test script can generate different test cases by binding different test data.

By using the keyword driven test, the test script is separated into many independent parts. Each part can be maintained separately without affecting the others. Thus the reusability and maintainability of the test scripts is improved.

### 3.2.5. Advantages of keyword driven test

The study case in this thesis adopts the keyword driven test as a test method. The advantages of using this method are listed below.

1. In the keyword driven test, the amount of scripts varies with the testing scale, but not with the number of test cases. The number of test cases can be increased by using different keywords instead of adding more testing codes.

2. The keyword driven test reduces the cost of script maintenance and speeds up the test case implementation [12].

3. The keyword driven test provides an extra layer between the testing code structure and implementation. When the testing tool or platform is changed, the tester does not need to
change the testing script design but only needs to rewrite the code for keyword implementation.

4. The tester can create a new test case by just using the existing keywords. It reduces the programming skill requirements of the tester [12]. A tester who does not know testing script implementation can still perform function testing.

The keyword driven test separates the testing data, testing logic and the testing script. With this testing approach, the modification and implementation of test cases becomes easier. The tester can modify a test case by replacing some keywords or changing the existing keyword script. It makes the testing script structure clearer and improves its reusability.

3.2.6. Disadvantages of keyword driven test

The keyword driven test is of course not perfect. It still has some issues that need to be solved, but it provides a direction for studying GUI testing automation. The disadvantages of the keyword driven test are listed below.

1. It needs a lot of investment at the beginning stage of the test automation, because a comprehensive keyword library needs to be built.

2. The quality of the keyword library is the determining factor of the test automation quality. Therefore, there are very high requirements for the flexibility and stability of the keyword driven testing framework. It increases the difficulty of keyword driven test implementation.

3. The tester who implements the testing framework needs to have good testing experience and programming skills [8].
Chapter 4

Considerations about test automation

4.1. Advantages of test automation

Some advantages of testing automation are obvious, others are not. The typical advantages of test automation are listed below.

1. Reduce test time and improve test efficiency.

   Compared with manual test execution, automated testing avoids the delay of human operation. The time difference gap becomes huge when the number of test cases is large. Test automation can save 25% to 50% [13] of the time spent on executing the same test suite. Therefore, more test cases can be performed by test automation within a limited period of time. The company can complete the software test in a shorter time and release the new version sooner. Manual testing requires more time, which means that defects may not be found until quite late in the process. The later the defects are found, the higher the costs of fixing them [14].

2. Improve test coverage.

   It is easy for test automation to reuse the test script for similar test cases, which should be run in different environments. For manual tests, test cases cannot be reused to benefit other test cases, even if they are quite similar. But with automated testing, the test case script can be used for a large number of similar test cases which only have a few differences. In this way, the test coverage is easily expanded.

3. Perform tests which cannot be performed manually.

   With the help of the test framework and by using the test script, automated tests can perform various kinds of test cases that cannot be performed by a human tester. For example, stress tests, load tests and crash tests etc.

4. Provide standardized testing process.

   Unlike human testers, an automated test can perform exactly the same test every time. It includes the same test input, the same execution steps and the same test environment etc. So the automated test has better consistency and repeatability. When a new bug is found, it is sometime
not easy for the human tester to repeat it and show it to the developer. But automated tests can be repeated as many times as necessary and generate the same test result. In addition, the status of AUT can be monitored and exported as testing data during the test execution process. The testing data can be used for application debugging, data analyzing and testing report generation [15].

5. Improve the working efficiency of testers.

Tests automation can release the tester from tedious and repetitive manual testing. Especially for GUI testing, the testing procedure from the tester’s point of view is just endless clicking and typing. This kind of test makes the tester feels bored. As a result, it is easy to make mistakes. With an automated test, the tester can save time on the physical labor input and focus on the test design of the new application. It is a better way to use the tester’s skills.

4.2. Disadvantages of automated tests

Although we already mentioned the many benefits of automated testing, it is not perfect for all kinds of tests. Some of the advantages are limited. If we are not aware of the limitations, we may have unrealistic expectations of the tests automation. Insufficient preparation for the complexity and difficulty of tests automation may waste time or even cause the whole project to fail.

1. Not everything is suitable for automation.

A computer is not omnipotent. Some testing tasks are easy for humans to perform but very complex for a computer to complete. To automate this kind of test cases will cost much more than to test it manually by humans. In addition, the verification may be not intelligent enough to detect defects like humans would. This kind of test case is not suitable for automation. Which kinds of test should be automated, and which should not, will be discussed in chapter 4.3 and in chapter 4.4.

2. Automation is only part of the test procedure.

Nowadays, most testing automation of focus just consists in to automating the test execution. The automation of test case generation is not common. Most tools only support the automation of test execution. There are still many testing tasks that need to be performed manually by the tester. To automate these tasks will increase the cost of the testing project, and sometimes the benefit is not significant. The tester should figure out the balance between automation coverage and project cost.

3. High requirement of tester’s skill and experience.

Compared with manual tests, tests automation has higher requirements on the tester. The tester needs to have test automation experience in order to implement automatic tests efficiently. The tester also needs to be familiar with the common features of the test automation framework and the language which is used to create the test script.

4. Test automation is not good at detecting new bugs [8].

Some defects were detected on the first run of automatic testing. Some defects were even detected during the script creation. Once the script is put into use, it is helpless to spot new
defects of the product. The test script is used repeatedly, just to make sure that the designed test cases are still passed. An automatic test rather takes the role of protector than detector. It helps the tester to be sure that the software works well.

5. Investment in automation may be huge [16].

Software testing automation is like software development. It means there are costs for testing script development and maintenance. For a company that has no test automation experience, the initial investment in a test automation project may be huge. It includes the time consumption and labor cost of testing tool selection, testing environment building, testing library building and the first version of testing script implementation. If the company decides to use a commercial testing tool, there is also an annual license fee to pay. When the test cases are automated, it is not the end of the test automation project but the beginning. The testing system should be maintained and updated according to the changes of the tested application. If the revenue of automatic testing is less than the revenue of manual testing or even the project investment, automation will become meaningless.

4.3. What kinds of tests are suitable for automation?

1. Regression tests.

In most cases, regression testing is the main reason to automate test cases, especially for GUI testing. Before the application releases a new version, the regression test is run to ensure that the changes do not affect the performance and features which the previous version guaranteed. For testing automation, the same test script can be reused. The more times the test script is reused, the more revenue is gained from test automation project.

2. Tests that cannot be performed by humans.

Beside performance testing, the GUI function testing also has many test cases which cannot be performed by humans. For example, some test cases require multiple testing operations simultaneously. Manual testing cannot guarantee that test actions are performed at exactly the same time.

3. Test cases which contain repeating operation steps.

Repeating the testing steps makes the testing process tedious and boring. The tester can easily become distracted and make mistakes. Compared with humans, computers are good at repeating tasks. A computer can complete the testing tasks quickly without any incorrect operations.


A monkey test is to input random operations and check the performance of the application. It is similar to stress testing. Monkey testing is not included in this thesis.
4.4. What kinds of tests are not suitable for automation?

1. The test cases which will not be performed as regression tests.

If the test cases just need to be performed a few times, testing automation will be not worth it. As the test automation design and implementation also need a lot of time and work, the more times the test script execution is repeated, the lower cost of each test and the more return on investment for the project.

2. GUI is changed frequently.

If the UI is always changed before the next round of testing, it means that the GUI is changed frequently. The testing script includes many identities of interface objects. The changes of UI bring about many changes to the testing script. This means a huge maintenance workload and more costs for the test automation project. A frequently changed UI reduces the return on investment and makes automation work cost more than manual testing.

3. Test related to user experience.

There are many user experience tests that cannot be performed by computers, or are very complex to automate. For example, to test whether the UI object appears in a correct format on different kinds of devices or whether the operation flow of the application feels smooth for the user. Currently manual testing is the best way to go for these tests.

4. The test needs fuzzy verification.

There are some bugs that humans can notice easily but may be ignored by machines. Compared to humans, the computer is still not intelligent enough. Most tasks that computers currently perform are those tasks in which each step comes with clear instruction, and the biggest advantage of a computer is that it can repeat the testing steps again and again without any error. Therefore, sometime manual testing is irreplaceable.
Chapter 5

Selection of testing tools

5.1. Overview

Automatic testing means to use a script to simulate the interaction between the tester and the software. The test tool has many features which already implement part of the needed functions, so the tester can just use these functions to build the test case quicker and easier. The more features the tool has, the less test script the tester needs to create and the faster the test case is built. In order to implement automatic tests quickly and efficiently, it is necessary to select an appropriate GUI testing tool. It is important to evaluate the various GUI testing tools on the current market and to choose the appropriate tool according to the actual situation of the project.

5.2. Our mistakes in the tool selection

Selecting the wrong tool can cause problems for the test automation project. Sometime, it is the factor which may directly or indirectly cause the failure of the project [17]. We made several mistakes for tool selection in this project. Inappropriate tools had been chosen and used to build test automation framework. Later on we realized the chosen tool was not the right for this project and had to start from the beginning again. This delayed the schedule of the whole project and increased the investment. Luckily we corrected our mistakes on time and processed the right tool selection procedure for the tools which are listed in this thesis.

The following list shows some mistakes that were made during the tool selection process and the summary of all the lessons that we learned. Some mistakes are common for the new GUI test automation projects. It is important that the tester is aware of these traps and avoids them during the tool selection procedure. This is the main purpose that we listed our mistakes and experience as below.

1. To not consider tool selection and evaluation as a complex and important procedure at the beginning of the project was a big mistake. Actually, the tool selection for a medium-sized organization will typically take from four to six person-weeks of effort, and may involve three to
ten people [8]. For a company that has no test automation experience and no test tools in use, insufficient preparation of tool selection can cause the failure of the whole project.

2. Not realizing that the tool selection is a time consuming process and that it is hard to follow a schedule. The time cost is also part of the project budget. Someone may spend several months to choose a tool and later find it is no good. Then the process needs to start over again. Some commercial tools have a one-month trial period. But it is not always enough to make the final decision. Studying the tools should start even before the tester gets the trial period. Most of the tools’ documents are available on the companies’ web pages. It is good to go through the whole tool documents before applying for trial usage. Otherwise it will lead to the inefficient usage of the tool trial time.

3. The requirements of the test tool were not decided before the beginning of tool searching process. The work of searching for the tool starts at the beginning of the project. At the moment there is only one requirement: the tool needs to be a GUI test tool for the Qt application. Due to the lack of tool requirements, many immature test tools were included in the scope of the evaluation. This wasted quite a lot of time. More details of the tool requirements will be discussed in chapter 5.4.

4. The unknown test automation plan led to uncertainty about the features needed for the project. All the brochures of these commercial tools announce that their tool is the best choice and can successfully implement test automation. Since the details of the test automation project were unclear when the tool search started, it was hard to decide which feature was mandatory for the project and which feature was optional. The project may either have ended up spending too much money on buying unnecessary features, or bought tools which do not have enough features.

5. The automation tool is another software application. To evaluate software is a complex procedure. The lack of knowledge of evaluation criteria for GUI automation tool leads to wrong choices. The evaluation criteria of the GUI test tool will be discussed in chapter 5.4.

The software manufacturing company usually provides software engineering serviced for different kinds of applications. Different applications may need different testing tools. Thus, the company may need other test tools in the future. The lessons we learned from the tool selection process can be important experiences for the tool selection of other projects. This project is considered as an investment in the training for tool selection and in how to start test automation project.

5.3. Tool selection process

There are several steps to complete the tool selection process. The following figure shows all the steps to find a proper testing tool.
The first step of the tool selection is the automation solution design. Get to know the test automation methods, and then know what kind of tool the project needs. In this step, the project time schedule and budget are also completed. The time put in the tool selection includes tool pre-study, tool training and tool trial. Making a business case means estimating the costs and comparing them with the project budget [8]. The costs do not only include the price of the tool license and training but also include the labor cost during the tool selection period.

The second step is to define the project needs [18]. Get to know which feature of testing tool the project needs and which feature the project does not need. Make the optimal choice according to the tool budget which is defined in the first phase. Once the requirements of the testing tool are clear, the first phase of tool selection starts. The testing tools which do not have these needed features are filtered out. The testing tools which have these features remain for the second phase of the tool comparison.

The tool trial is the most important step of the tool selection process. This step aims to become familiar with the tools and check their features. Simple test cases can be used to become familiar with the tools, and some special test cases can be used to check the features which the project needs. For most commercial tools, there are usage demos and technical support.

5.4. Tool comparison in the first phase

According to the tool selection procedure already discussed, the initial requirements of testing tools for this project are listed below. Several testing tools are selected and compared with each other in terms of each requirement. This is the first stage of the selection process. This comparison will narrow down the list of selected testing tools. [18] The second stage comparison will evaluate the testing tools for more detailed features. The selection procedure may take several months, but the trial period of commercial tools is usually one month. Therefore the first stage focuses on the document study and the second stage focuses on the practical usage.

The following sections are about the requirements of the testing tool. There are also comparisons of all the tools for each listed requirement.
5.4.1. Supports the Qt application test

The basic and essential requirement for the testing tool of this project is to support testing of the Qt application.

“Qt is a full development framework with tools designed to streamline the creation of applications and user interfaces for desktop, embedded, and mobile platforms. With Qt, you can reuse code efficiently to target multiple platforms with one code base.” [19]

It was a very popular software development framework, especially when Nokia still marketed Meego as the main platform for Nokia phone. Nowadays, many people still use Qt, but the number is less than for these very popular software development frameworks, for example Java or Android. The market situation of the software development framework also affects the testing tool. Since Qt is not so widely used, many well-known testing tools do not support Qt, for example Rational Functional Tester [20]. A tester who has experience with the other kinds of GUI testing, like Java UI testing or web UI testing, may still need to find a new testing tool for a Qt test automation project. Luckily the test automation experience will still benefit the usage of the new testing tool. If only the testing tool supports the GUI toolkit, the testing methods and implementation do not differ. It means that after the tool selection, no matter whether the project is about Java test automation or Qt test automation, or the tester chooses the A tool or the B tool, the implementation is the same. More about this will be explained in chapter 5.6.2.

Due to the current market situation of the Qt development framework, there is not much mature testing tool support for QT UI testing. At least there are not many free testing tools that can be used. Unless Qt becomes the mainly used software development framework, not many people are interested in Qt testing tools and rather devote their time to the development and maintenance work of open source tools which support Qt testing.

Here is a list of GUI test automation tools which support the Qt application:

- Squish [21]
  Squish is developed by Froglogic GmbH which is located in Germany. It is the core product of this company. Besides testing tools, the company also provides consulting services for Qt related tests automation. Squish is very famous for Qt tests automation. The Qt official website recommended two Qt testing tools. They were KD Executor and Squish [22]. But nowadays the development of KD Executor has come to halt. And even KD Executor recommends people to use Squish for Qt tests automation [23].

- Ranorex [24] and TestComplete [25]
  Compared to Squish, where the main selling point is tests automation for Qt applications, Ranorex is more of a generic testing tool. It uses the Qt Accessibility module which is part of the Qt GUI toolkit to identify the Qt UI components. [26] Ranorex is a commercial testing automation tool which is developed by Ranorex GmbH. The company also provides consulting services and training services for software testing.
  As a testing automation tool, TestComplete is quite similar to Ranorex. Therefore, it will not be compared in this thesis.
Testability Driver [27]

Testability Driver (TDriver) is an open-source project which was developed by Nokia. After Nokia abandoned Qt, the development and maintenance work of TDriver was also stopped. Nokia’s development removed all the TDriver related content from their webpage and server. Nowadays, it is available in Gitorious as an open-source project. [27] Before it was abandoned by Nokia, TDriver was the most popular free GUI testing automation tool for the Qt application.

Dogtail [28]

Dogtail is a GUI test tool and test automation framework written in Python. It uses Accessibility (A11Y1) technologies to communicate with desktop applications. A Software Assistive Technology Service Provider Interface (AT-SPI) is commonly used with Dogtail to provide accessibility [29]. QT AT-SPI was available in 2012. And with the help of QT AT-SPI, Dogtail 0.8.0 has started supporting QT tests automation [28].

The five testing tools listed above are tools which, according to statements by clients or companies, have been used in practical projects. There are some other tools which also declare to support Qt testing but have no practical usage examples. Those tools are not included in the comparison. This is not the only reason that some tools are eliminated from the selection list. Many tools are not complete GUI testing frameworks. Some tools only have Record and replay functions, for example Hooq [30] and Open HMI Tester [31]. Some only have programming libraries of accessing to the Qt UI objects. To build a real testing framework with these tools will take a lot of time and effort. It is almost a project of testing tool development. It needs a developing team to accomplish this task. Testers should focus on test case design and implementation, not on testing tool development.

5.4.2. Testing diversification

Qt is a cross-platform software development framework. Qt applications are developed on various kinds of software and hardware platforms. For desktop applications, Qt supports Windows OS, Linux OS and Apple OS. Qt also supports multiple mobile platforms. Before Nokia gave up Symbian and Meego OS, Qt was mainly used to develop applications for these mobile platforms. Nowadays, Qt supports Android, iOS, QNX and embedded Linux [32]. Therefore, the testing tool must be able to perform GUI testing automation for the application, which is running in multiple environments. Although this thesis focuses on GUI test automation of the Qt application, a company may also have some other UI testing tasks for which the applications use different GUI toolkits. For example, a company may have web applications that need to be tested. So it is better the tool also supports some other types of UI besides Qt.

Out of all the testing tools for Qt testing, Squish is the tool which supports the largest number of comprehensive platforms. It supports most of the platforms which support the Qt application. Only a few platforms are not supported, for example Blackberry. The wide support of platforms is one of the reasons that make Squish an outstanding Qt testing tool. Squish also supports many other kinds of GUI tests, for example Java Swing/AWT.

As a commercial testing tool, Ranorex also supports various types of GUI tests. Its official website lists more than a hundred supported GUI technologies [24]. It includes the major GUI technologies of desktops, mobiles and web platforms. The biggest shortcoming is that Ranorex does not support the test for Linux or UNIX operation systems. Unlike Squish, which is advertised as a Qt
testing tool, the selling point of Ranorex is its diversification test for “any technology”. But of course this advantage becomes useless for a Linux application test.

TDriver is the testing tool which targets Qt application testing. It supports the test for Qt desktop application on Windows, Mac and Linux. It was the main testing tool for Symbian and Meego applications. Since these two mobile platforms have been abandoned by Nokia, the development of TDriver was stopped. The number of platforms supported by TDriver is decreasing. Especially for the current popular mobile platforms, Android and iOS, platform migration may need much efforts.

Dogtail is a testing tool for Linux desktop applications. It is a generic testing framework which supports tests for all AT-SPI supported GUI toolkits [28]. It does not support any other platforms except Linux.

5.4.3. Script languages

The script which is used to implement test cases should be a common language, [33] or be easy to learn. If the tester is familiar with the testing script, it will save a lot of time for programming training. On the contrary, if the script which is used by testing tool cannot be learnt in a short period, it will not only delay the tool evaluation process, but also bring troubles for the script maintenance [17]. The problems will become obvious when a new tester joins the project. Therefore it is better that the testing tool supports multiple script languages. Then there is a better chance that one of the scripts is known to the tester.

The follow table lists script languages which are supported by different testing tools.

<table>
<thead>
<tr>
<th>Testing Tool</th>
<th>Supported Script Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squish</td>
<td>Python, JavaScript, Ruby, Perl or Tcl</td>
</tr>
<tr>
<td>Ranorex</td>
<td>C# and VB.NET</td>
</tr>
<tr>
<td>TDriver</td>
<td>Ruby</td>
</tr>
<tr>
<td>Dogtail</td>
<td>Python</td>
</tr>
</tbody>
</table>

Table 1 Script languages which are supported by the testing tools [21] [24] [27] [28]

As table 1 shows, Squish again shows outstanding performance in the aspect of supported script language. Robust test automation for Windows applications is one of the selling points for Ranorex, therefore C# and VB.NET are used to provide flexibility and integration for the team who has year of experience of testing automation for Microsoft .NET applications. Although Ranorex does not support python, the tester can still use python to access the Ranorex project which uses C# or VB.NET. TDriver was written in Ruby, and its supported script language is limited to Ruby. It is the same situation for Dogtail, which is implemented in python. These two scripts are very common languages for tests automation and they are easy to learn.

5.4.4. Tool related materials and technical support

Tool related materials and technical support of the testing tool is very important for both tool evaluation and the practical test automation project. During the test automation procedure, the testing team will face various kinds of issues, especially for the testing team which has little test automation experience or starts using a completely new testing tool. If there are not enough technical
documentation, powerful technical support of the testing tool usage and an active tool community, a lot of time will be spent on tool studying and troubleshooting. It makes the project schedule hard to follow and the costs hard to control.

In this aspect, Squish and Ranorex show a big advantage as mature commercial tools. Compared with free open source testing tools, the commercial tools are very good in the following areas

- **User manual**

  Both Squish and Ranorex have User guides and API documents. The User guide describes all the tool features in detail and how to implement different kinds of GUI tests automation with these features. Squish is even better than Ranorex in the aspect of technical document. Unlike Squish, the API document of Ranorex does not have code examples for API usage explanation. This is especially inconvenient for the new user. TDriver and Dogtail have no feature explanation documents. Since TDriver was transferred from http://developer.nokia.com/ to Gitorious, the only document available is the simple installation instructions. Even the API manual has disappeared. An old version API manual of TDriver can be found in http://www.rubydoc.info/. Although Dogtail has its own website, only simple installation instructions and API manual can be found on that website. A good thing is that the API document will be updated when Dogtail is updated. The bad thing is that the API document does not have any example code.

- **Usage example**

  Squish and Ranorex have also published many technical documents. Some articles explain the practical issues of tool usage from the user’s point of view. Some articles show the customer’s story for a test automation solution. With these technical articles, the user will have a deep understanding of how to use the tool to achieve their goals. Besides technical documents, Squish and Ranorex also have various kinds of example codes to demonstrate the tools’ usage in practice. The long customer lists on their webpages show the tools’ successful usage in real business project. In contrast, the practical usage information that the user can get from the webpages of TDriver and Dogtail is limited. The example code of TDriver is a very simple case of a calculator test. That does not support problem solving when the users use the tool in their own project. Luckily, for both TDriver and Dogtail, there are some blogs on the Internet to show how the tools are used to match their testing projects [34] [35]. These blogs were written by individual testers. Most of the code examples are still at the simple interaction level. Although the testers recommend the tools, they admit that they are still far away from advanced use [34].

- **Training**

  Both Squish and Ranorex have a series of training videos. In addition, they provide online webinars about the use of the tool and GUI test automation theory. The tester can sign up to free webinars on different topics. Since TDriver and Dogtail are free software, the testers are supposed to learn everything by themselves. The tools do not provide any training.

- **Technical expert support**

  All the commercial tools provide technical expert support. The customers always have countless questions when they use the tool for their own project. They can get most of the answers from the support team by email or phone. Squish also provides chargeable customization and consulting services [36]. Both Squish and Ranorex have forums for discussion. The users can share their user experience there and also put forward new questions from their own projects.
Experts from the testing tool companies are also active in the forums and in answering the questions. Neither TDriver nor Dogtail have a dedicated forum. There is also no technical team support. As TDriver is a dead project, fewer and fewer testers use it. So there are not many topics related to TDriver on the Internet. Dogtail is a little better than TDriver. There are some discussions about general questions of Dogtail and AT-SPI. But there are quite few discussions about the Qt test with Dogtail since the Qt AT-SPI has only recently been made available. The tester can also not expect that someone who knows the tool very well has time to answer questions. The truth of free testing tools is that most of the questions will not be answered and the testers should figure out the answer by themselves.

5.4.5. Cost

The cost is an important consideration of testing tool measurement. It does not only include the money paid for the tool license, but also includes the labor cost and the investment in usage tainting. Comparing the commercial tools with the open source tools, the cost are reflected in different fields. The main expense of the commercial tools is the license fee. Squish is the most expensive of all the testing tools. One of its named user licenses for the Squish edition version is 7,200 [37] Euros. The license contains 12 months up-front usage which allows three testers to perform testing on Windows. As a comparison, the Ranorex group license is from 690 Euros to 3,490 Euro [38]. Squish’s high license fee is the only factor which makes the tester hesitate during the tool selection.

As open source software, TDriver and Dogtail do not need licenses. But due to the immaturity of the testing tools and the lack of technical documentation and support, the up-front investment in those tools is incalculable. The time spent on tool installation and environment setting up may be huge. The tester may also need to debug or even modify the testing tool to make it work for the current Qt testing projects. If TDriver is chosen, the company also needs developers who are responsible for the tool’s updating and maintenance. TDriver has barely been updated in two years, whereas the Qt software development framework is updated continually. It is almost another project of software development. The situation of Dogtail is a little better than TDriver, it is continually being updated. However the updating interval is a bit long. The latest update of QT AT-SPI was two years ago.

There is also a difference in the labor cost between commercial tools and open source tools after the tool is put to use. The advantages of commercial tools are, among others, high stability, easy usability, systematic training and powerful support. This guarantees that the tester can spend less time on becoming familiar with the tools. It means that when a project changes the tester, the new one can start to work quickly without a long period of study.

5.4.6. Ease of use

As mentioned in the discussion above, the ease of use affects the costs of the project. It is also important for the time schedule of the project. If the tool is very hard to use, it will not only delay the project schedule, but also foil the tester’s confidence, especially for the first test automation project.

A powerful IDE with user-oriented graphical interface is the main factor that affects the usability of the tool. IDE usually includes many tools to simplify the creation and organization of test cases. A browser which shows the structure of the test suite may be included in IDE. It makes the management of test cases very intuitive. Test cases can be dragged and dropped directly to change
the construction of the whole test suite. The IDE also includes a test script debugger. The test automation implementation is like software development. The testing script has data structure, variables, functions etc. The tool’s debugger can help the tester find errors quickly. With the help of IDE, the tester can execute the script step by step and check the status of the components for every step.

Similar to other items evaluated before, commercial tools are better than open source tools in this regard. Ranorex targets the testing of Windows applications, it integrates the open source .NET IDE SharpDevelop [39]. The testing script can also be debugged in Visual Studio. So for the tester who is familiar with .NET IDE and Visual Studio, Ranorex can be used immediately without training. Of course this benefit becomes useless for the test on Linux. Squish integrates the popular IDE Eclipse. No matter which operation system the test cases are running on, the tester can always use similar IDEs for testing script implementation and maintenance. Squish is the only tool that has mature IDE for all operating systems. Neither TDriver nor Dogtail have IDE, they only includes the driver to run pure Ruby or Python scripts.

IDE also provides many other features which ease the tester’s work. For example, it identifies the graphical component of AUT, generates testing report etc. More details of these features will be discussed in the second phase of the tool evaluation. Most of these features can also be approached with the tool’s APIs. Comparing Squish with Ranorex, the latter has more features implemented by means of the graphical user interface, and Squish prefers to use APIs to drive the functions. For the tester without much experience with GUI tests automation or coding, IDE with a graphical interface is more acceptable. It eases the learning tasks for the beginner in test automation or testing tools. But the graphical interface decreases the flexibility of the testing tool. To accomplish complex testing, the tester eventually needs to rely on the APIs rather than the graphical interface of IDE.

5.5. Tool comparison in the second phase

After the first phase of the comparison, the tool list is narrowed down. As shown in the tool selection process, the selected tools need to be checked as to how they work in practice. The GUI testing tools have many similar features. Their advertising brochures of course state that their features are perfect. But with further in-depth research, the difference between the similar features of the testing tools will be discovered [17].

The features evaluated in this chapter are indispensable functions for building the basic element of keyword driven testing framework. These features are used to create many small modules which formed the foundation of keyword driven testing framework. The feature comparison in this chapter is also an explanation of how to create the bottom layer of keyword driven testing framework.

5.5.1. General architecture of testing tool

The four testing tools compared in the first phase represent different types of testing tools which are currently available on the market. Squish is the famous product which highlights the test for a certain GUI toolkit; it is not commonly used for testing projects of other software development frameworks. Ranorex is a generic commercial test tool. It is not a dedicated testing tool for a certain software development framework, on the contrary, the main selling point of Ranorex is the wide support of GUI toolkits. TDriver is an obsolete testing tool while Dogtail is still an active testing tool.
development project. Both of them are open source and free. But TDriver was initiated by a company while Dogtail is maintained by an open source community. Although these testing tools are so different, they have one thing in common, they all have a similar software structure.

The architecture of the testing tool contains some modules which are the engines of the basic features. All the evaluated testing tools have a similar software construction which includes similar modules, and that leads to similar testing features. This may confuse the customer for the tool selection. This is why the second phase of tool evaluation is necessary. Rather than checking what feature the tool has and what feature the tool does not have, the second phase evaluates each similar module of the tool and finds out the differences between them.

In order to compare each module of the testing tool, the tester first needs to understand the basic structure of a testing tool. Figure 5 shows the basic structure of a testing tool. The following structure is created mainly based on the tool structure of Squish and TDriver [40] [41].

![Testing Tool Architecture](image)

**Figure 5** Testing tool Architecture [40] [41]

There are two basic modules in Figure 5. Actually, most tools contain more modules than that, but these two modules constitute the mandatory parts of a testing tool. They are the basic features that a testing tool must have in order to simulate the interaction between user and software application.

The UI inspector on the right of the Figure 5 is a detector and executor. It works as the eyes and hands of the tester that remember all the things shown on the screen when the software application is running. It remembers all the information like what UI objects the software has, where they are located, and what the UI looks like and so on. This UI information is necessary for the test script, otherwise the tester script does not know what kind of interface it can manipulate.

This UI object information is passed to the script execution driver on the left of Figure 5. The script driver works as the brain of the tester. It contains various libraries which can help to build the testing script. The scripts driver run the testing script and interprets the scripts into the testing actions, like text typing, mouse clicking and dragging etc. The information of the testing actions is sent back to the UI inspector which can perform these testing operations like the tester did in manual tests.
Both the UI inspector and script execution driver in Figure 5 have a communication component which is in charge of the information exchange between these two modules. The communication components of the two modules transfer the information by using the method of client and server. The UI inspector works as the server, which provides the UI information, and the script execution driver works as the client, which handles the received UI information. For some testing tools, the two modules are executed in two separate processes and communicate with each other over a network connection. The benefit of having independent modules is that if the AUT crashes during the test, the UI inspector will crash too since it hooks the AUT together, but the script execution driver will not crash and can record the situation or restart the test again automatically [40].

For this kind of tool architecture, it is possible that the two modules are installed in different machines separately. Squish and TDriver have quite similar tool architecture. The whole test tool can be installed in the same machine on which the AUT is running, or the script execution driver and UI inspector are set up on the different machines so that the tester can execute remote testing. Ranorex and Dogtail cannot perform remote testing without the help of a remote execution tool. If the tester wants to run a test which is installed on another computer, the tester either logs into the testing machine remotely or copies the whole test case execution files to his own computer. The Ranorex mobile test is an exception. Ranorex only has a separate UI inspector for mobile devices. The mobile device which installs the UI inspector package can connect to the tester’s computer by cable or Wi-Fi and works just like a remote test [42].

The UI inspector module and script execution driver module will be explained in detail in the following chapters.

5.5.2. UI Object identification

UI identification is a key feature of the GUI testing tool. Test automation is the procedure of simulating the interactions between user and software application. In order to operate the software UI components, the testing script needs to recognize them. For example, if one testing action is to click a login button, the first step of creating this action with a testing script is to find where the login button is. To operate the software UI only according to the coordinate position is not reliable, since the coordinate value may change on different devices. Hard coding the coordinate value into the testing script will also bring a lot of problems for the script maintenance. Just like for a manual test, the interface of the AUT should be interpreted into a set of components with descriptive information for each one of them.

The UI inspector module in Figure 5 is in charge of recognizing the UI objects and collecting the UI information. The UI inspector attaches to the AUT and communicates with AUT by using the accessibility technology. Figure 6 below shows the accessibility architecture between AUT and the UI inspector.
To make the application accessible, the application should register to the UI inspector so that the UI inspector can know there is a software interface which needs to be fetched. Generally, commercial testing tools like Squish and Ranorex use the configure file to register the application. For TDriver, the tester needs to manually add additional parameters to the command which runs the application. Dogtail is a little different from other tools. When the assistive technologies of Linux OS are enabled from the system configuration, Dogtail can access any application on the desktop.

Once the application is testable, the UI inspector will dynamically load various accessibility libraries for UI information fetching. The different accessibility libraries are used for different GUI toolkits. As mentioned before, this test automation project targets Qt GUI test. Therefore, the testing tool should contain the Qt accessibility library which has the ability to recognize the GUI object that is created by the Qt software development framework. Except TDriver, the other three testing tools have multiple accessibility libraries, for example the Java accessibility library. There are also accessibility libraries which are used to access operation system components. Linux uses AT-SPI as the interface of the accessibility library and Window uses Microsoft UI Automation (UIA). Other operation systems also have their own accessibility technology which the testing tool may adopt for UI identification. This is why some operating systems are supported by testing tool and some are not. Different kinds of accessibility technologies are interoperated by the UI inspector and work together to get the necessary UI information.

The accessibility library also works with other function modules to accomplish the UI identification task. Event monitor is a module which is used to monitor the UI changes and report these changes to the accessibility library [45]. So if the interface is changed during the test, for example a new window pops up after the login button was clicked, the accessibility library will check the application interface again and update the UI information.
5.5.3. The information of UI objects

What UI information does the accessibility library get and pass to the script execution driver? Figure 7 is an example to show the fetched UI information.

![Figure 7 UI object identification](image)

The first image on the top of Figure 7 is the interface of a student information management system. The second image on the bottom of Figure 7 is a UI information visualizer which is used to represent the information of the UI object in the way that the tester can understand. Thus the tester could know what UI information the UI inspector gets. It usually has a view to show the hierarchy of the UI objects, a view to show all the properties of a particular UI object with their corresponding values and a view to show the graphic appearance of this UI object. The UI visualizer tool included in the Ranorex tool suite is called Ranorex Spy. The tool is not a mandatory function for the testing script creation, but almost every GUI testing tool has this feature. Usually, the UI visualizer is part of the tool IDE and works as the assistant of the tester during the test script creation. Since TDriver and Dogtail have no IDE, this feature is built as a separate tool. It is called Visualizer for TDriver and AT-SPI Browser for Dogtail.

The Ranorex Spy in Figure 7 shows the information of the UI object which is marked with a red line. Here is the list of common things that UI object information may contain:

- Property and its value. A UI object always has a set of properties. Each property has its own value. The set of property values constitute the state of this UI object. These properties are used to describe the UI object. For example, there may be a property with the UI object’s length or width. For a window or a button, there may be a property with the text or title of the UI object. There are some common properties which almost every UI object contains. For
example, like the property which indicates the type of UI object, the red marked object in Figure 7 will include the property-value pair such as “type=textbox”. Although the name of the property and its value are different for different tools, the meaning that they represent is the same. Another common property is the visual state of the UI object. It shows whether the UI object is available or not. For example a button may have a property-value pair like “visible='1'” to show that the button is available and not grey.

- Dependency. The UI objects of the application interface are organized into a tree structure [46]. Usually the biggest window of the displayed interface is the root note and other widgets are located on the different levels of this UI component architecture. As Figure 7 shows, the textbox is under a widget where the value of property “accessiblename” is “Input Student Info”, and the widget is under a window where the value of property “title” is “Student Info Manage”. There is an additional layer (the container[7]) between the widget and the textbox. It is not a UI component, it is created to bind the 7th textbox with its corresponding label “native” and distinguish it from other textboxes.

- Behavior method. Every UI object can be operated in one or multiple ways, like select textbox, type in textbox, click button and so on. When Ranorex Spy shows the actions belonging to the UI object, it means that these actions are available for this object. The test script can only manipulate the UI object with the operation that is available for it.

All the UI object information above is encapsulated into one set which is used to uniquely locate the UI object. Missing object information may cause misrecognition, or fail to recognize the UI object. Commercial tools do a better job than the free tools in this aspect. All the tools can handle the simple interface very well. But for these complex UI objects, the capability of testing tools is different. Squish can be configured to access the custom UI object [40]. It is used in case that the UI object is not created by the standard GUI toolkit. And with Ranorex, the tester can build the test script based on image [47]. It is used for an UI object which is hard to access to get the property value. For these objects, Ranorex can take a snapshot of the UI object and use that image as the object identity. Neither TDriver nor Dogtail have any mechanism to prevent a situation in which the UI object cannot be recognized. For Dogtail, users report that in some cases they have problems in recognizing the custom UI object [48]. Since the updates for TDriver have almost ceased, there is the possibility that even some standard UI objects cannot be recognized by TDriver when the GUI toolkit of AUT is updated to a newer version.

5.5.4. UI object name

To know the UI object’s identity is an important task of test script creation. Once the UI object is identified, the test script can refer it as an individual object for testing operations. The UI inspector delivers the UI information to the script execution driver, but how the UI information can be used by the test script is still a question. Generally speaking, there are two ways that the UI object is referred to in the test script.
One way of indicating the UI object is to give it a symbolic name [40]. In this way, the test script can call the UI object’s name and manipulate it directly. The name is usually a string and it is created like the ID of the UI object. Two textboxes in the same window may have many quite similar attribute values, but they will have a unique symbolic name. For example, for the textbox in Figure 7, it is named as “text” and another textbox in the same window is named like “text1” for distinction. And when the tester wants to click this textbox but not the “birthday” textbox above it, the test action can be “focus ‘text’ ”. The symbolic name is generated automatically when the UI inspector fetches the UI object information. Dogtail has problems in generating the symbolic name of the UI object which is created dynamically when the application is running. For this kind of UI object, Dogtail can get all the properties’ information but the object name is shown as “unknown”.

Another way of UI object expression is to pick up several specific properties and combine them into one unit [40]. This unit will be used as the identity of this UI object. Thus the format of object ID is a set of property-value pairs. For example, ID of the textbox in Figure 7 is shown as

\( /\text{form[@title='Student Info Manage']}/\text{container[@accessibleName='Input Student Info']}/\text{container[@]/text[@accessibleRole='Text']} \)

The formats of the property-value pair in the unit may vary depending on the testing tool. The unit which is shown above is what the Ranorex ID looks like. The syntax of how these property-value pairs are organized in the unit is designed based on XPath [49]. For other testing tools, the common format of the object ID is a brace or parentheses containing a list of property-value pairs which are separated by a comma or space. Here is an example of Squish object ID, it represents a push button.

\{text='Yes' type='QPushButton' unnamed='1' visible='1' window=':Address Book - Delete QMessageBox'}

As shown above, the value of the property “window” is the symbolic name of another UI object. Actually, the property’s value can be of many formats. It can be a string, a number, a variable and even a regex for some tools. The purpose of the multiple kinds of value formats is to increase the flexibility of object matching, so that the complex testing scenario could be built up by a simple and effective test script. It is also the reason that the tester cannot always use symbolic names in the test script. Symbolic names cannot provide a range of property values for object matching. And that limits the possibility of script structure. For example, if there is the same testing action for hundreds of UI objects, the test script has to list all these symbolic names and that produces a lot of redundancy for the test script. For both Squish and Ranorex, the wildcard can also be used to set up the property’s value [40] [49]. Instead of searching the exact matched object, it only limits part of the value in case of the value is changed. For example, a window of the text editor may have the “title” property where its value was “text editor - new” when the application began to run and later became “text editor - saved”. The matching value in test script can be “text editor*” to include all the possible title names [40]. TDriver and Dogtail have less flexibility in property values. The wildcard cannot be used directly in the property-value pair.

5.5.5. UI object mapping repository

All these object names are organized in an object mapping repository. In this repository, the two format IDs of the same UI object map to each other. The tester can use either one of the format IDs to call the UI object. If the tester uses the symbolic name, once the object is referred in the test script, the script driver will search for the name from the object mapping repository and get the necessary property values of the object. The values are used to match the actual graphic object on the interface.
Only Squish and Ranorex have an individual file for the object mapping repository. The tester can also edit this file. It means the tester can edit the UI object name to any string, as long as the name is unique. In this way, the symbolic name can be defined to describe the UI object in a way that makes it easier to know what object it is for the tester. More importantly, the editable object mapping repository makes the script maintenance easier. When the property value of a UI object is changed in a new release version, for example the title of a window changes from “Login” to “User verification”, the object ID cannot match the exact graphic object any more. In this case, all the places of the test script that refer to this object need to be modified. It may cause a huge amount of script maintenance work. But if the tester uses the symbolic name of the object, only the property value in the object mapping repository needs to be modified [40]. The symbolic name can still remain the same. Otherwise the tester needs to catch UI information again to get the new name of this object and modify every place which uses the old symbolic name. In a big project, the UI re-capturing and script modification may lead to lots of time waste and unexpected errors.

There are other nice features that the commercial tools have for the management of the object mapping repository. For Squish and Ranorex, the tester can edit the rules that decide what kind of property-value pairs are picked for object matching. So the tester can make a customized syntax for object ID definition. And for Ranorex, the tester can create folders to sort the related object IDs together for better organization.

There are two main ways through which the tester can see the information in the object mapping repository. One way is to use the recorder to record the test cases. When the test case is recording, the script driver will collect the UI object information and generate the object mapping repository automatically. Meanwhile the symbolic names of the UI objects are also shown in the recorded script. It is the only way that the object mapping repository is created automatically. Otherwise the tester needs to add the object information one by one. It is also impossible to build the object mapping repository manually, due to the huge number of UI objects belonging to an application. Therefore, the first step of the script creation of a new automation project is to record the test case, or just trigger every UI object on the interface. This is the most important use of the Record and replay method. It is also the reason that although the Record and replay method is not considered an efficient test methodology, it is still a necessary feature for the testing tool. For those tools that have no record function, the tester may have problems with UI identification. Another way is to use the object visualizer tool which is similar to the Ranorex Spy in Figure 7. All the properties and their corresponding values belonging to a single object are shown on the tool, the symbolic name of the UI object is also shown. By using this tool, the tester can check any UI object information from the screen.

All the testing tools which are evaluated in this thesis have recorder and UI visualizer functions and their working mechanisms are similar. But commercial tools have more functions, which makes UI identification easier for the tester. The open source recorders and UI visualizers only have very basic features.

5.5.6. Test script execution platform

The script execution driver in Figure 5 provides testing libraries for test case implementation. The tester can use various kinds of APIs which are offered by the testing tool to automate every step of the manual testing. The critical testing steps which need to be implemented are the simulation of the
interaction between the tester and AUT and the generation of the testing report. To accomplish these steps, the following APIs are frequently used:

- The interaction of the mouse and keyboard, like clicking and typing. Except for Dogtail, which cannot be used for mobile device testing, all other testing tools also have APIs to implement the actions of human gestures.

- APIs which are used for verification. In general, there are two ways to verify the test. The first one is to compare the property value of the UI object. The second one is to do the bitmap comparison for the image which is shown on the screen or the screenshot of the whole interface [50].

- APIs for monitoring whether the UI object which will be operated is available or not. This kind of APIs are used to imitate the tester’s checking every time before performing any testing action. For example checking whether a window has popped up before clicking any button in that window.

- APIs for setting how long it will take to perform a testing action or how long it needs to wait before certain UI object is available.

- APIs for exception handling. Uncertainties may interfere and cause the test automation to crash. To prevent that, the test script needs to specify how to handle unexpected situations, for example an unwanted window popping up, so that the tester does not need to check the test automation before it finishes running [33].

- APIs for testing report generation. The testing report is normally used to show the each case’s test result and record the running status of test cases. The report may include the explanation of testing steps, the error or warning messages, statistics analysis of the whole test suite, screenshots of the failure and so on. Almost every testing tool has these kinds of APIs discussed above. But the commercial tools have more functions for each category of APIs than the free tools. Take the verification as an example, TDriver has no API to perform bitmap comparison; Dogtail can only take a screenshot of the desktop screen and compare that with a given image; Squish provides a mask feature which can choose a part of screenshots for the comparison and improve the possibility of matching [40]. And beside the mask, Ranorex even offer a preprocessing feature for the image. The image can be captured not only by taking a screenshot but also by choosing the UI object as a picture. The captured image can be preprocessed by the various options. Ranorex’s preprocessing options are “DownSize”, “Edges”, “EdgesSobel”, “GrayScale” and “Threshold” [47]. Ranorex has two matching methods to compare the preprocessed image with the given image. One is to check whether the given image includes the preprocessed image, and another is to check whether the preprocessed image is the exact same as the given image.

The script execution driver of Squish and Ranorex is more like a testing platform. It contains various modules which provide additional features beside the ones discussed above, such as the interface to access an external database etc. The test automation running on Dogtail is just like other
python projects which use the python development package. The tool provides basic functions to implement the core testing steps, and the tester needs to implement the additional features with python script. TDriver’s situation is similar to Dogtail. Some APIs’ functions on TDriver even have the testing specific parameters and Ruby parameters mixed together.

5.6. Issues to consider in the tool selection

5.6.1. Commercial tool vs. free open source tool

The two phases of tool comparisons show that there are major differences between commercial testing tools and open source testing tools. This triggers an important question before selecting a testing tool. Commercial tool or open source tool, which kind of tool is good for the test automation project?

Although the open source tools are not as good as the commercial tools, they still account for a large proportion of the testing tool market. The main reason is that open source testing tools are free. It is especially important for those test automation projects which are still in the study stage and may not have any budget at all. But no license fee does not mean no costs. As discussed in the previous chapters, commercial tools are much better than open source tools in all aspects. Open source tools are developed and maintained by volunteers. Unlike the development team of a company, the loose organization of the open source community can guarantee neither an updating schedule nor software quality. To make an open source tool work, there is a long way to go for the tester. And the cost of filling in the quality gap between a commercial tool and an open source tool may be more than the license fee of the commercial tool. But if a software engineer who has test automation experience is available, many companies would still like to choose an open source tool, especially a company that has a tradition of providing contributions to the open source community. After all, the open source tool already has the basic tool architecture and core module implemented. Modifying the existing testing tool needs much less work than building a totally new testing tool.

The free open source tool is a better choice for the project under the following conditions:

- The test automation project is scheduled for a long period. The studying and development of a testing tool requires at least several months depending on the experience of the tool developer. Therefore, the test automation project should be scheduled for more than one year. The company should not look for immediate return on its investment. The benefit of choosing an open source tool is that it will cost less and less with the progress of the project. The development of an open source tool only needs a big investment at the beginning. After that the developer will have a deep understanding of the testing tool and spend less time on updating and maintaining the tool. While for a commercial tool, one needs to pay the license fee every year.

- The test automation project is to test single types of applications which are the main products of the company. Without paying a lot of money to buy an omnipotent tool, the company can develop a tool which is quite fit for their own test automation project. The tool’s features are all designed based on the requirements of the test project. Another benefit is the test
automation project is independent from commercial tool manufacturers. If the price of a commercial tool is increased or the tool company gives up the tool, it will not affect the test automation project.

- The company should have a strictly organized development team for the development of the open source tool. Although the tool already has the main modules implemented, the modification of the open source tool is still an independent project. It needs to be organized as another software development project. And the team leader should make sure the changes by the developer will not cause the project to fail.

The commercial tool is a better choice for the project which is under the following conditions:

- The test automation project needs to start quickly and the company expects return on investment in a short period. The company which provides software engineering service may get out-sourcing testing tasks from other companies. This kind of test automation project usually has a tight schedule and there is very little time for tool preparation. In this case, it is unrealistic to use an open source tool.

- The test automation project includes testing tasks for various kinds of applications. The tester may need to build the test suite for desktop testing, web testing and mobile testing. Due to the shortage of technical documentation and support, the installation and environment configuration of an open source tool is a nightmare. The tool is vulnerable and crashes easily. To make the open source tool work on different platforms and environments will cost more time than estimated.

- The test automation project needs to perform various kinds of tests automation. The test cases may include monkey test, remote test, and regression test etc. To handle complex testing scenarios, the testing tools should have many features. The implementation of some features needs a lot of time if the company chooses the open source tool.

5.6.2. Making a decision

There are many options to select the testing tool for the Qt application test.

For the purchase of a commercial tool: if the company does not have too many test tasks for Linux embedded software, Ranorex is a better choice. It is a robust and mature testing tool which is much cheaper than Squish. The company can retain Linux embedded software tests as manual tests, and use Ranorex to have high quality tests automation for all other platforms. Ranorex can be a compromise choice between a commercial tool and an open source tool. Especially for a company without much test automation experience, Ranorex is a very good starting point. With the technical support, the company can gain test automation experience for their projects in a short period and also reduce the investment risk.

If the main testing task of the company is to test Linux embedded software, then Squish is the only available tool for serious testing of Qt Linux embedded software.
For open source tool development, TDriver is a good choice. It was given good feedback before Nokia dropped the project. And it was the first choice of the Qt software testing tool among all the open source testing tools. TDriver supports the tests on various operating systems, and it can be integrated with other testing frameworks. For example, it can be integrated with Cucumber to perform agile testing; it can also be integrated with TEMA to perform model based testing [51].

There is a third option besides buying a commercial tool or developing an open source tool. If the company has been using a testing tool for a long time, but that tool does not support Qt software testing, an add-on can be created for the testing tool to make the tool recognize the Qt UI object. Or the company can pay the tool manufacturing company to do that, which will be much cheaper than buying a new testing tool. The add-on is similar to the Qt accessibility library in the UI inspector. Qt software development framework provides several accessibility classes which can be used together with the operating system accessibility technology to make the Qt application accessible. The benefit of this option is that the tester is already familiar with the tool and the company can save a lot of money and time in tool study and training.

The testing tool abstracts the complex interface into the high-level tree model. No matter which way the test script chooses to refer the UI object, the object identity does not include any information related to the GUI toolkit that the AUT used. For example, when the testing action is to click a button, the test script does not need to know whether it is a Qt button or a Java button. It means that the testing script does not include any information about which GUI toolkit is used to create the interface. The testing script does not include any information related to the testing tool either. Although the UI object format may vary from tool to tool, the meaning is the same. None of the UI identities include information about testing tools. All information of the GUI toolkit and testing platform is filtered out from the UI identity. Therefore, no matter which GUI toolkit is used to develop the AUT or which testing tool is chosen for the test automation project, the test automation implementation is the same. It is why the content of the use case in chapter six will not be affected by the decision made in this chapter.
Chapter 6

Case study

This chapter demonstrates a whole test automation implementation procedure. Firstly, a test case design procedure will be introduced. Then, the most important content for this project, a keyword driven testing framework is built for test case generation purpose. The bottom layer of the testing framework is developed by the testing tool. Chapter five explains how the bottom layer is created. Other layers of the testing framework are created base on the concept of keyword driven testing method which is introduced in chapter three. The testing scripts implemented in this chapter are structured into many small units. The unit is called keyword, and keyword layer of the testing framework implemented in this chapter is the collection of these units.

6.1. Introduction of AUT

A student information management application (SIMA) from the open source software community is selected as the study case [52]. SIMA is used to collect and manage students’ information. It has four main windows. One is the login window. To login with different roles will have different authorities. With the login as student, the user can only check students’ information. With the login as administrator, the user can check, add or modify the students’ information. SIMA connects to a database which is used to store these students’ information. The database information is divided into four categories; they are personal information, studying status, student reward record and student disciplinary record. There is a combo box in the login window. If the user chooses to login as a student, the combo box will be grey. If the user chooses to login as an administrator, the combo box will become available and the user can specify which categories of information will be added after the login. By choosing different functions in the combo box, the administrator will login in different information management widgets. There are five functions in total. The first four functions are used to add different kinds of new student information. The fifth function is used to check and modify existing student information. Figure 8 shows the login window with the available combo box options.
There are five widgets according to the five functions in the login window. After each login, only one widget is available depending on the administrator’s choice. Four widgets are located in the second main window which is the Student Info window. Each widget is used to add one kind of new information. For example, if the administrator wants to add the basic information of a new student, the first item of the combo box needs to be chosen when the administrator logs in. After login, the widget in the upper left corner of the Student Info window is available, and the other three widgets remain grey. Figure 9 shows the Student Info window where the “Input Student info” widget is available.

If the intention of the administrator is to check or modify existing student information instead of adding new information, the user can choose the fifth item on the combo box when logging in. Then
the Student Info Search & Modify window appears. The forth main window is the Student information checking window which is used by students to check their information.

6.2. Test case design

Test cases are the foundation of testing script. And test case design is the first step of the testing process. Test case design does not start after the completion of software development. As shown in the V-model which is introduced in chapter two, test case design starts at the beginning of a software development project [8]. Figure 10 shows how the product specification is created at the beginning stage of a project. It is drawn base on my working experience.

![Figure 10 Product specification generation process](image)

The first step of a software project is to accomplish the product specification according to the customer’s requirements. The product specification explains the initial design of this application and which features will be eventually implemented. The application’s UI and software architecture will be designed according to the product specification. The feedback generated during the process of product design also affects the product specification. Not only UI designers and software developers contribute to the initial model of the application, testers also participate to make sure that the software design completely meets the customer’s requirements. The UI will be verified so that it matches the function design. At this stage, the application’s GUI test case design already starts in accordance with the product specification. The testing design step also checks the logic of application functions and provides feedback to improve the product specification.

The following test cases are designed to verify two features of AUT. One is the login function, the other is the function of adding a new student information to the database.
<table>
<thead>
<tr>
<th>Testing feature</th>
<th>Test case number</th>
<th>Test operation</th>
<th>Expected test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>TC 1</td>
<td>Login MISA with wrong user name and password</td>
<td>Login failed</td>
</tr>
<tr>
<td>Add new student’s basic information</td>
<td>TC 2</td>
<td>Login MISA with function “Input student info”, add a group of information for a new student</td>
<td>New information is added successfully</td>
</tr>
<tr>
<td>Add new student’s basic information</td>
<td>TC 3</td>
<td>Login MISA with function “Input student info”, add a group of information for a new student but some fields’ information are missing</td>
<td>User get warning message</td>
</tr>
<tr>
<td>Add new student’s basic information</td>
<td>TC 4</td>
<td>Login MISA with function “Input student info”, add a group of information for an existing student</td>
<td>User get error message, information cannot be added to database</td>
</tr>
</tbody>
</table>

**Table 2** Document information of manual test case

The first step of adding new student information is the successful login. So TC2, TC3 and TC4 all include the verification of successful login. In this case, there is no need to make a separated test case of successful login. A test case about failed login is enough for the testing of login function.

There are more than four implemented test cases in the whole test suite. Due to the similar implementation, some other test cases are actually implemented but not explained in this thesis. For example, in order to test the function of adding student’s reward information, the administrator chooses “Input reward” instead of “Input student info” when logging in, there are also other three test cases which are similar to TC2, TC3 and TC4. In this thesis, only four test cases are picked to demonstrate the GUI test automation process which uses the keyword driven method.

The test cases shown above may be OK for manual testing but cannot be used directly for automatic testing, because the test script cannot perform some common actions like the tester. For example, knowing the user and password is enough for the tester to perform the login action without any instructions. But for the automatic test, there should be a test script to implement the exact operation of every single testing step. And all these testing steps should be well documented. Otherwise it may cause trouble for the test script maintenance, especially when the tester is replaced. Take TC2 as an example, the documented information is shown as below. The information can be documented in the testing files, or be coded into the testing log and generated as the testing report.
Test Case SIMA-002: Add new student information

Summary:
test the function that the administrator can add new student information

Steps:
1) Launch the application
2) Click the combo box of the ‘user type’, and select “admin”
3) Click the textbox of ‘username’, and type in “admin”
4) Click the textbox of ‘password’, and type in “123”
5) Click the combo box of the ‘choose model’, and select “Input student info”
6) Click ‘login’ button
7) Click the textbox of ‘student no’, and type in “123”
8) Click the textbox of ‘name’, and type in “Jenny”
9) Click the textbox of ‘gender’, and type in “female”
10) Click the textbox of ‘class no’, and type in “c118”
11) Click the textbox of ‘dept no’, and type in “x51”
12) Click the textbox of ‘birthday’, and type in “1986.04.03”
13) Click the textbox of ‘native’, and type in “Sweden”
14) Click ‘Submit’ button

Expected Results:
A new small window pops up with the text “submit successful” on it.

Table 3 Document information of test automation case

Usually, the company has already been using manual testing for their software applications for quite a long time, and the purpose of the test automation project is to automate these manual test cases. So the company will not spend too much time on the test case design. The tester normally just modifies the manual test document and uses it for automatic testing.

However, if the company decides to use automatic testing at the beginning of a new development project, the test case design method they can use may be totally different. The test automation project can use the Model-based testing approach to create the test case automatically. Model-based testing is test automation methodology which abstracts the behavior of AUT into various models and generates the test cases based on the models [53]. If this testing approach is adopted, not only the testing actions are automated but also the test case is designed automatically. There are some other famous testing methodologies, for example Test-driven development (TDD) and Behavior-driven development (BDD). TDD and BDD first design the test cases and start implementing the software application based on the test cases [54]. They are commonly used for agile software development.
6.3. Testing framework

Test automation does not just mean to automate the test case step by step. If it did, the tester could just use the testing tool to record all the manual test cases and replay them when needed. The test automation project is similar to software development project. It usually contains many testing files, and the number of files will increase along with the increased number of test cases. When the interface of AUT is changed and some test cases need to be modified, the maintenance of the testing scripts becomes complex. Therefore, the testing script should be written as maintainable clean code such as that of a software application.

The GUI test automation framework is built as a foundation of a test automation project. It provides a reusable testing script for the implementation of different test cases. The testing framework is made up from various modules which provide different functions that the test case needs. Some modules are provided by the testing tool and some modules are created by the tester. It gives the testing framework a clear organizational structure which helps the tester to create new test cases and modify the existing test cases easily and quickly. The testing framework is normally built based on a testing solution, for example a keyword driven testing framework is built for this study case. For keyword driven testing framework, the testing modules are organized into different layers. Each layer takes the lower layer as the structural unit and can be used to support the implementation of the upper layer. The tester can modify each layer separately.

The beginning task of the test automation project is to build the test framework to support further testing. The testing framework is designed and implemented in four [55] layers. The following figure shows the structure of the GUI testing framework

![Testing framework architecture](image)

Figure 11 Testing framework architecture

6.3.1. Function Layer

The layer in the bottom of Figure 11 is the testing function. It implements the smallest element of the whole test automation project and all other upper layers are created based on this layer. Each testing action is created as single function in this layer. For example, the four test cases which are shown in table 2 all include the testing action that selects which information management widget the user would like to login to. The widget selection can be implemented into an independent testing function. As discussed in chapter five, UI object ID and the APIs which are provided by the testing
tool are commonly used in the testing script to simulate manual testing operations. Therefore, the API module and UI object repository module are the mandatory parts for the building of the function layer.

In manual tests, the tester may need to repeat certain actions in many test cases, like the previous example of widget selection which is included in at least four test cases. However in an automatic test, the test action can be written as a shared function and each test case just needs to call the shared function to perform similar testing actions. In the example of widget selection, the selected option for each test case may be different. To make the shared function usable for different situations, the testing function can include a variable to represent the selected option. For different test cases, the variable can be assigned a specific value which means the corresponding widget selection. This avoids much of the redundant code. The shared functions can still be used for creating new test cases.

Function layer is a set of testing functions, and each one includes various variables. The variable may be the UI object which the testing action needs to manipulate to. Like the previous example of widget selection, the listed items of the combo box are implemented as the variables of the shared function. In the UI object repository, the listed item contains a property-value pair which the property is called “accessablename”. To implement the shared function, the corresponding value of property “accessablename” is created as a variable instead of a string. In this way, five listed items in the combo box are combined into one variable. When the widget selection function is called, the variable is assigned with different strings which refer to the different listed items of the combo box.

The variable of the testing function can also be the expected result value which is used in the verification function. Take TC2, TC3 and TC4 as an example, after the tester finishes typing the student information and submits the data, a small window pops up to show whether the data has been stored successfully or not in the database. These three test cases all include the comparison action to check whether the pop up window is the correct window. The comparison action can be implemented as a verification function. In the function, the title name of the pop up window is compared with the expected value. The expected value differs for different test cases. TC2 is supposed to have a window to show that the information is submitted successfully. Therefore, the window’s title should match a string with text “submit”. By the same token, the expected value should be “Submit warning” in TC3 and “Submit error” in TC4. To make the verification function a shared function for multiple test cases, the expected value is implemented as a variable, so that it can be assigned various kinds of strings according to the testing logic.

6.3.2. Keyword layer

Keyword layer is the collection of keywords for this test automation project. Keyword is actually the encapsulation of a series of testing steps. It is created to implement the testing process which is considered as one testing operation according to the way of human thinking in manual testing. As discussed before, each testing step is implemented as an individual function in the function layer. Thus, various testing functions which are related to each other in logic aspects will be classified into one keyword. A keyword implementation is an individual code block. It uses the testing function of the lower layer to compose a small part of the testing process. The name of the keyword usually describes what that small part of the process does.

Part of the TC2 testing process is to fill in the basic student information. To do that, the tester needs to click one textbox and type in the proper value, then move to the next textbox and repeat the
There are seven textboxes on the “Input student info” widget, so the clicking and typing actions need to be repeated seven times. As discussed before, similar clicking and typing actions can be implemented as one function in the function layer of the testing framework. Likewise, the series of repeated testing actions can be arranged together and implemented into one keyword in the keyword layer. The keyword can be called “AddStuInfo” which explains what this part of testing process does. For this keyword, the clicking and typing function in the function layer is used as the smallest element for the keyword implementation. It consists of a loop execution function which calls for repeating same clicking and typing function seven times. If the test case is about entering disciplinary record of a student, the series testing actions can also be implemented as another keyword which is called “AddDiscipRecord”. Unlike the keyword “AddStuInfo”, the keyword “AddDiscipRecord” needs to use multiple testing functions, since the testing process also includes the action of combo box selection. In a word, the keyword layer is the abstraction of the testing function layer. It makes the testing script modular so that it is easier to be maintained and reused.

6.3.3. Test case layer

The second layer in Figure 11 is the test case implementation layer. It is called test case layer in this thesis. All the necessary keywords are built up in the keyword layer for the test project. They can be used by the test case layer to compose the main part of a test case. For example, the test cases TC2, TC3 and TC4 can be created by the following four keywords.

- Keyword “AppLaunch” which launches the AUT.
- Keyword “Login” which implements the testing steps in table 3 from step 1 to step 5. After this keyword is executed, the “Student info manage” window appears with the “Input Student Info” widget.
- Keyword “AddStuInfo” which implements the testing steps in table 3 from step 7 to step 14. After this keyword is executed, new student information is submitted to the database.
- Keyword “SubmitVerification” which implements the result verification part in table 3. In this keyword, the pop up window in the screen is compared with the expected result and the test case execution is considered as pass or fail.
- Keyword “Termination” which is not included in the test case design but is used to close all the open windows of the AUT and terminate the running application.

TC2, TC3 and TC4 consist of the same keywords. But the values of the variables in the testing functions which are included in the keywords may differ. For example, the input texts of the seven textboxes differ for different test cases, the expected result in the verification function is also different. The different values of the variables are collected in a testing data file. The keyword defines the logic of the test case, and the testing data describe the different scenarios with that testing logic. By using this method, the tester can build one test case but execute many test cases with different input data. The test cases which have the same testing logic are sorted into one category. Multiple groups of testing data are bound to this category. Adding a new test case can be achieved by just adding a new group of testing data without modifying the testing script.

The test case layer can just pick up the required keywords which are created in the keyword layer and arrange them according to the logic of the test case. Therefore, a test case is created by a sequence of keywords which are bound with a group of testing data. The function layer is the testing
function pool which provides all the functions needed to build up the keyword. The keyword layer becomes the keyword pool for test case generation.

6.3.4. Test suite layer

The layer at the top of Figure 11 is responsible for the test case management. It is called test suite layer in this thesis. The configuration of the whole test suite needs to be set up in this layer. Some configuration parameters, especially the environment parameters, should not be hardcoded into the test script. Thus, the testing tool usually provides a separate file which contains the configuration setting when the tester starts a new project. The configuration parameters include the script path, AUT path, the GUI toolkit of AUT and so on. The tester can modify the configuration file according to the practical situation of the testing project. When the test suite starts to run, the testing tool loads the configuration file and set up the parameters automatically.

6.3.5. Summary

Each layer in Figure 11 is the abstract of its lower layer. This kind of testing framework structure makes the test script maintenance and modification work easier. The tester can modify the separated layer without changing other layers. For example, a new test case can be created by rearranging the order of keywords and adjusting the testing data in the data file. And in this case, only the keyword layer is changed. Also if a testing action is changed, the tester only needs to modify the corresponding testing function in the function layer. All the keywords, test cases and test suites which include the modified testing function get updated automatically.

The structure which has separate layers also simplifies the requirements for the testers. The test automation project may involve various people who handle the different tasks and have different skills. Not all of them need programming skills and know every detail about the project. People with different roles can work on separate layers of the testing framework. The tester who has programming skills can be in charge of the creation of the testing functions and keywords. And the project manager who knows the project design can just use the keywords to construct the test suites without understanding the details of the testing script implementation. When the testing framework has been created, the testers who handle the maintenance work can still be divided into two categories. The function layer and keyword layer are considered as lower layers which can be maintained as other software development projects, and the other two layers which are considered as the higher layers can be maintained by a person who does not know much about coding but understands the testing theory very well, for example the tester who performed manual tests before. In this way, limited labor resources can be used effectively.

For the tester who handles the testing task in the upper layer, the test implementation concept is closer to the manual test. As table 2 shows, login is considered as one testing step in the manual test. But like as shown in table 3, clicking the button is a testing step. The test implementation concept in the upper layer is closer to human thought, whereas for the tester who handles the testing task in the lower layer, the test implementation concept is closer to software development. The main concern of the tester is how to build up maintainable testing functions.

The six test cases which are shown in table 2 are just examples to explain the testing framework building and project implementation procedure. Therefore, neither the testing function nor the keyword are frequently reused. In a realistic test automation project, the number of test cases is very
large. Thus, the benefit of testing framework will be more obvious in a large project. Different combinations of keywords with large testing data can easily produce huge numbers of test cases which can never be achieved by manual testing. Also the time cost for script maintenance and updating will be shorter.

6.4. Test implementation issues

The testing framework is the most important step of the test implementation procedure. It is the foundation of the test automation project. Large numbers of test cases can be created by using the testing framework. Besides testing framework building, there are several other issues which are also important for test implementation.

6.4.1. Comparison in real time

Every test case has at least one verification function to decide whether the test is passed or failed. The verification functions usually use the comparison APIs provided by the testing tool to compare the output with the expected value. Not all of the testing comparison operations are used to verify the result of the test case, some are used in the middle of test case execution to make sure that the test is running in the correct way and the UI is in the right state to continue testing. In a manual test, this kind of testing operation is not recorded in the test document, it may be not even noticed by the tester. But in a test automation, if the test script does not perform these operations, the machine will never do it automatically. Take TC2 as an example, before entering any text into the textbox, the tester should make sure that the textbox is empty. The tester checks this without thinking in a manual test. In test automation, there should be functions which could perform this check to ensure that the UI of AUT is ready for the next testing operation. There are two testing functions implemented in this example to check whether the textbox is empty before entering text. One is used to fetch the text of the textbox. Another is used to remove the existing text if the fetched string is not empty.

6.4.2. Handling of incidents

During the automatic test execution, an unexpected event may happen and cause the test case to crash or fail. For example, when AUT tries to store the new student information into the database in TC2, there may be a pop up window from the firewall to warn the computer that someone is accessing the database. In this case, the test script should know how to handle the incident and keep the test running. There is a testing function implemented in this example to close the unexpected dialog.

6.4.3. Time setting

Each UI object has different states. Any new state of an UI object usually becomes available after a certain time delay after the UI state was changed. For example, after the login button is clicked, it
needs a certain time for the window “student info” to pop up. If the correct window does not pop up within this time, the test case may fail. The limited time is the timer for UI objects. There are also timers for mouse and keyboard operations. The timers define the operation speed of mouse and keyboard. All these timers need to be set to proper values. Generally, the time intervals are generated automatically during the test recording and there is no need to change them. These timers can be the reference value for new script creation.

6.4.4. Pre- and post-processing

Testing functions for pre- and post-processing are needed for most of the test cases. The post-processing functions are used to eliminate the execution traces and set the state of AUT back to the initial stage, so that the execution of one test case will not affect the performance of the next test case [8]. For example, TC2 execution is finished after successfully adding data to the database. To prevent that the execution of TC2 will affect the execution of another test case, there are testing functions implemented in TC2 to close all the windows that remain on the screen, so that other test cases can start from the beginning.

The pre-processing function is used to set up the AUT to the right position before the test case is executed. It prepares the testing environment for test execution. For example, the purpose of TC4 is to test the feature that the administrator cannot add student information which already exists in the database. Therefore there should be operations which enter the same information as in TC4 before executing TC4.

6.4.5. Report of testing result

Test report creation is an important part of an automatic test process. A test report does not only show whether test cases are pass or fail, it also illustrates the test processing details which are very important for the tester to troubleshoot the failed test cases. Commercial testing tools usually automatically generate a test report, while open source testing tools only collect the log information which is implemented in the test script. The follow figure shows a testing report. The testing report contains the testing result of the four example test cases in Table 2. It was created in HTML format by Ranorex.
Figure 12 Testing report
6.5. Discussion

6.5.1. Analysis of testing report

The testing report mainly contains two parts’ information. The first part is some basic information of the whole testing process. It includes the testing device, testing operating system, the duration and so on. The report also shows pie chart of numbers of passed test cases and failed test cases. The duration of completing the test cases is 1.4 minutes. Manual test will take around 10 minutes to complete the same test cases. For the workload of a testing project, manual tests usually take several days. According to the timing scaling of this project, automatic tests can shorten the time to a night.

The second part shows the details of each test case. It lists every executed test case. Each test case includes three levels’ information.

Ranorex divides each test case into three parts. They are SETUP, MainOperation and TEARDOWN. Each part is composed by several keywords. The keywords are listed in the executed order. SETUP is the pre-processing of the test case, and TEARDOWN is the post-processing of the test case. Since TC2, TC3 and TC4 have the exact same login procedure, the keyword AppLaunch and Login were used to constitute SETUP; keyword AddStuInfo and SubmitVerification are used to constitute MainOperation; and keyword Termination is used for TEARDOWN. Every part can be bound with several groups of testing data. The test script is executed with one group of testing data, and then the execution is repeated with the next group of testing data. Thus, with three groups of testing data, TC2, TC3 and TC4 are executed by using the same keyword list. The current bound test data is shown under each iteration.

Under every keyword, the report lists each testing step which is executed in the keyword. All the log information of testing steps is created in the testing. Testing team can use the log information to know the details of the tests processing. In case there is failed test case. The testing report will not only list the log information of each step, but also show the screenshot of that failed testing step.

6.5.2. Implementation difference

As previously discussed, no matter which testing tool is selected for the test automation project, the testing framework of the test automation project is same. Due to the difference of testing tools’ features, the testing framework implementation is a little different, if other testing tool is selected instead of Ranorex.

After the keyword creation is complete, the tester can use the keywords to establish a test case directly without knowing the detailed script implementation. The main purpose of using a keyword driven test approach is to ease the work of test case creation and maintenance for the people who are not involved in programming related work. The string name of the keyword describes the testing operations which are implemented in the keyword. It is bound, together with its corresponding test data, in one data table. The data table can be used to generate and update the test cases. The way of using the data table to handle the testing operation is quite similar to the manual testing process.

The following table is the data table for TC2. As an example shown in chapter 6.3, TC2 is composed by five keywords. The text in the first column is the keywords, which are listed in the sequence of test execution. The text in other columns with dark color is the name of the variables.
and the text under the blue cell is the variable’s value. For example, the data “admin” is the value of “username” for Login keyword.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Argument 1</th>
<th>Argument 2</th>
<th>Argument 3</th>
<th>Argument 4</th>
<th>Argument 5</th>
<th>Argument 6</th>
<th>Argument 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppLaunch</td>
<td>App_path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stu_infor_system.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Login</td>
<td>Username</td>
<td>Password</td>
<td>Widget_selection</td>
<td>Login_error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>admin</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AddStuInfo</td>
<td>Student_No</td>
<td>Student_Name</td>
<td>Gender</td>
<td>Class_No</td>
<td>Department_No</td>
<td>Birthday</td>
<td>Nation</td>
</tr>
<tr>
<td></td>
<td>Han112</td>
<td>Hanna</td>
<td>F</td>
<td>c118</td>
<td>x51</td>
<td>1986/4/28</td>
<td>Sweden</td>
</tr>
<tr>
<td>SubmitVerification</td>
<td>Pop-up_Win</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Submit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Termination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4** Data table for TC2

The project manager or manual tester who cannot do the programming work can just use this table to maintain the existing test case or even create new test cases. The keyword name and testing data explain which testing steps are executed in the test case. If the tester wants to insert more testing operations, the test case can be modified by just adding more rows with the keyword which is implemented in the lower layers and its corresponding data.

The data table of a test case is usually stored in a data file. The format of the data file is normally XML, CVS or XLS. Table 4 is not the standard format of data table. The format of a data table varies from tool to tool. In Ranorex, table 4 is separated into two kinds of data table. One the table contains only variables and values. The other table contains only the keyword list; these two kinds of tables are linked together. For other testing tools, the data table can be created in the format which is the easiest to understand. Although the format is customizable, they are all quite similar to table 4. The content and usage of the data tables are the same.

The IDE of Ranorex provides a feature which can link the data table to the testing script. It can automatically search the corresponding script of the keyword and execute the script in accordance with the keyword order that is listed in table 4. When a keyword script is being executed, this keyword’s corresponding testing data in table 4 is also red for the script execution. Most of the testing tools do not have the feature described above. It requires the test script itself to implement these features. Therefore, if other testing tools are selected, there will be a separate script file to implement the function which converts the test case table into executable testing script [40]. The function will map the keyword string name into actual keyword script, and read the data to get the value of variables for their keyword. The implemented script of the table converter can be reused for other keyword driven test projects. The following figure shows the usage of the table converter in a keyword driven test solution.
As shown in figure 13, the files of a keyword driven test automation project are usually divided into three categories. One category is the implementation script files of keyword and testing functions which are created in the lower layer of the testing framework. Another category is the data files which contain the list of keywords and bound data to build up the test case design. These files are created in the higher layer of the testing framework. These two categories are also handled by people who have different roles in the test automation project. The third category is the implementation script of the table converter. It is the glue which links all the kinds of files together. When the test project starts to run, it first loads the project configuration file to set up the environment. Then both the data file and the script of the table converter are loaded. According to the testing data, the table converter drives the specific script in the lower layers of the testing framework to complete the test execution. All the log information will be collected together to generate the testing report.
Chapter 7

Conclusion and future work

The early stage of a test automation project is important. A wrong decision made at the beginning may lead to failure of the whole project. This thesis mainly focuses on solving the problems which are faced by a company with no test automation experience but that intends to start a new test automation project for their products. GUI testing automation is a comprehensive subject that the testing solution may differ in accordance with the tested application. The main work of this thesis is to provide a general solution for a GUI test automation. With the research in this thesis, a company gains test automation experience which can be used for various GUI test projects in the future. The research focuses specifically on project preparation and initial implementation. Besides the subjects covered in this thesis, there are still many topics related to GUI testing automation which are worth studying in the future.

Agile testing is widely adopted by many test automation projects. BDD is a software development method used for agile software development and testing. The keyword driven test method which is discussed in this thesis can be combined with the BDD method for software development projects. The design and implementation of the testing framework in this thesis is based on the situation that the project is built to automate manual test cases. However, agile testing is becoming more and more popular, and the testing method in which BDD is combined with a keyword driven test may be a trend for GUI tests automation.

Cucumber is a testing tool which is used for the BDD method. It can be integrated with the Qt testing tool to perform GUI tests automation in agile software development projects for Qt applications. Since Cucumber is written in Ruby, it can be easily integrated with TDriver. TDriver already has a testing example which uses both TDriver and Cucumber. As discussed in chapter five, the tester may need to do some additional work to make TDriver usable for their project. How to develop and maintain a testing tool may be a good topic for a company that is more interested in open source software.

The thesis evaluates several testing tools. The comparison mainly focuses on the desktop software tests. Web UI tests can be another interesting topic for a study of GUI testing automation. All the commercial tools in this thesis can perform web UI tests. Commercial web testing tools can again be compared with free testing tools. The process of comparing web UI testing tools is almost the same as the process of comparing Qt testing tools. There is a popular free web UI testing tool called Selenium. Unlike Qt GUI tests, Selenium is a mature testing tool which has already been used in real business projects.
Another potential subject for future study is a practical case study of web UI tests. The implementation method of a web UI test is somewhat different from the study case shown in this thesis.
References


