



Bachelor Thesis Project

GUI driven End to End Regression testing with Selenium



Author: Christer Hamberg
Supervisor: Johan Hagelbäck
Semester: VT 2017
Subject: Computer Science

Abstract

The report shall begin with a summary, called abstract. The abstract shall not be longer than a paragraph, and is not divided into more than one piece. It shall contain:

- A short background description to the area of your project
- A description of your research problem
- A motivation why this problem is interesting to investigate
- What you have done to answer the problem
- A short summary of your results

From reading the abstract the reader should clearly understand what the report is all about. The purpose of the abstract is to make the reader interested in continue reading the report, if it covers something that the reader wants to know more about.

Keywords: fill in some keywords for your work here. Examples:
software architectures, adaptive systems, network intrusion detection, ...

Preface

You can have a preface in the report if you want, but it is not necessary. In this you can write more personal reflections on your thesis project. In the preface you can also take the opportunity to thank the people who have been particularly helpful during the report writing, for example if you had any contact with a company that helped with the project, people that guided or helped you during the project, or your family and friends that supported you during the project. The preface shall not be longer than half a page.

Contents

1	Introduction	5
1.1	Background	5
1.2	Previous research	8
1.3	Problem formulation	10
1.4	Motivation	11
1.5	Research Question	11
1.6	Scope/Limitation	12
1.7	Target group	12
1.8	Outline	13
2	Method	14
2.1	Scientific Approach	14
2.2	Method Description	14
2.3	Reliability and Validity	18
2.4	Ethical Considerations	18
3	Implementation	19
3.1	Components of FIA	20
4	Results	27
4.1	Test case selection	29
4.2	Test case execution times	31
4.3	Flaky test results	35
4.4	Modifications to GUI navigation	37
5	Analysis	39
5.1	Required execution times	39
5.3	Keeping test cases running	40
6	Discussion	40
6.1	Flaky test results	41
6.2	Flaky test results	41
6.3	GUI navigation requiring modifications	45
7	Conclusion	46
7.1	Future Research	46
	References	47
A	Appendix 1	48

1 Introduction

In this chapter you shall give an introduction to your thesis project. It shall start with a broad overview of what your project is all about. Similar to the abstract, the introduction shall make the reader interested in continue reading your report. Don't be too detailed here; there are plenty of opportunities to add details in later chapters.

< FIX all reference links to the IEEE material , check with Johan how to add links to all the different methods etc. That chapter needs a second look, it is not good enough in its current format!!!>

In July 1969 a man stepped on the moon for the first time. Almost 50 years later with all the technology advances available, we still struggle to run regression testing of interactive web applications in a successful and cost efficient way.

The repetitive task of regression testing can be one of the costliest testing activities performed in the software development project. The same test cases are repeatedly being executed in order to ensure that introduced code changes do not break anything in the application. Due to the frequency and repetitive nature of the task the execution depends heavily on automation of test execution in order to achieve both speed and cost efficiency in the regression test activity.

Automated testing of single-page applications (SPA) is hard to achieve as test cases regularly break, test execution takes long time and test cases can result in flaky results. As a result of these problems testing is often split in different parts, which are tested separately from each other.

This thesis studies the problems observed during regression testing of a commercially available web application and proposes solutions on how to overcome these issues, with the intention to avoid splitting up the testing of the different components separately from each other.

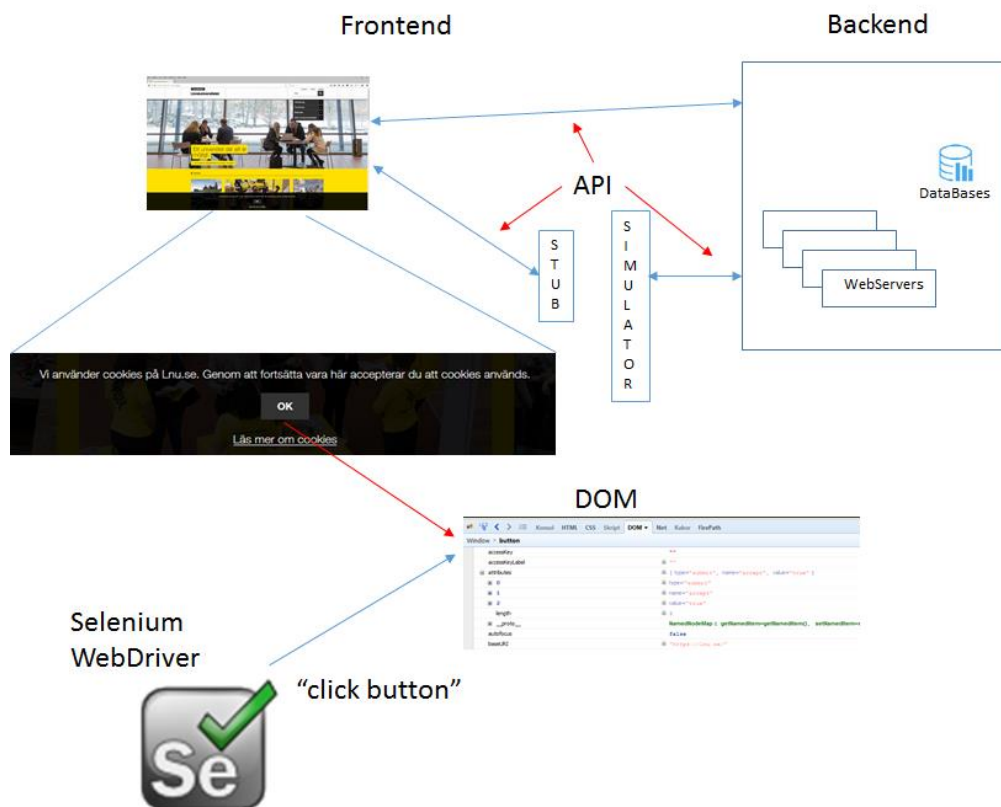
1.1 Background

After you have described your project, you shall continue with writing a background to the area your project is in. Here you describe theories necessary to understand your project and explain terms you will use in the report.

Example: if you do a project that is about evaluating software architectures, you describe what a software architecture is, why it is important to design an architecture that suits a specific software system, methods for evaluating and comparing different architectures, etc.

An SPA is logically divided into two different entities [Fig 1.1], the frontend and the backend. The backend implements the business logic needed by the system with for example file/database access to retrieve data needed for the realization of the functions.

The frontend implements the graphical user interface, used by the user to interact with the system. In an SPA the GUI is realized as a web page running in an ordinary web browser. The communication between the frontend and backend is done over an application programming interface (API). Which enables the frontend to send and receive requests and data to/from the backend. SPAs implemented today are implementing the same kind of functionalities as ordinary desktop applications are, and could be seen as ordinary desktop applications.



[Fig 1.1] Test Automation of SPA

Regression testing

Testing is done in order to build confidence that the software works as intended. When a piece of code is modified, regression test is performed and as described in the IEEE definition a selected subset of test cases is executed.

However regression testing is not done in order to test that the code change functions as intended. Instead regression testing focuses on making sure that the changed code did not break any existing functionality.

Development of a software application is an iterative process, causing numerous software changes on already tested and approved code. These iterations cause a need for retesting of the code, regression testing. Due to the repetitive nature of the regression test activity [REF xxx] it is one of the most expensive test activities, consuming upto 80% of the testing budget [REF yyyy].

The IEEE definition of regression testing is *Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.*

Interactive web applications

An interactive web application is in most cases logically separated in two parts. A backend that implements the functional/business logic of the application and a frontend which implements the graphical user interface (GUI) that the user uses to interact with the system.

The logical separation is needed and used for security reasons, because the frontend does not have direct access to persistent data storage.

Backend

The backend system in a web application implements the business logic. It also provides the application with the needed access to persistent data storage, often implemented using some kind of database.

Frontend

The frontend implements the GUI needed for the user to interact with the web application. A web browser is ordinarily used for the access to the application. However the web browser itself does not implement the logic of the GUI. What it does is providing the infrastructure needed for a single-page application (SPA) to run in.

API

An application programming interface (API) is used for sending of commands and data between the frontend and backend.

STUB and Simulator

A STUB and/or Simulator is used when testing the components separately from each other. They act as the “other” entity and responds to API requests in a predefined manner.

Document Object Model

The web browser keeps track of the web elements implemented by the SPA in an internal register, document object model (DOM). Different components such as buttons, text fields etc. that are available in the GUI, the locations and states of them are stored in the DOM.

Selenium WebDriver

Selenium WebDriver is an open source application that provides a programmable interface to interact with the web browser. The user can control various web elements over this interface, such as clicking buttons, entering text in text fields or validate results etc. I.e. automatically perform the tasks a user would do while using the web application.

The web browser stores the available web components in an internal register (DOM). It is the DOM that Selenium WebDriver interacts with to do the requested tasks.

The regression test activity of an SPA

Regression testing of an SPA is often divided into 3 different activities.

1. Testing of the frontend (GUI) using various techniques, where both the look and feel of the application is tested, as well as how to navigate, usability and clarity in the information presented.
2. Test of the backend (business logic) is normally done by triggering the API requests that the GUI would do. However as this is done without using the GUI, the problems with the navigation and slowness of the application running in the browser are avoided.
3. A limited test suite consisting of some test cases that tests that the two entities (frontend and backend) can communicate with each other and some scenarios are tested.

1.2 Previous research

Here you briefly describe what others have done in the field or how others have attempted to explain or solve the same or similar problem as you are investigating. It is okay to refer to tech articles and online blogs and portals, but you must also refer to published articles in the field. To find related articles, use the search tools listed [here](#).

This chapter is not good enough it needs to be rewritten and enhanced

Testing is an area that has been studied and researched for a longer time. The testing area is also extremely wide [xxxx], and the scope covered in research includes both the used testing methodology, test case coverage, selection and reduction, tools, Graphical User Interface testing etc.

Most of the researched areas have a common goal to increase the efficiency in testing and ultimately reduce costs. Another factor driving the research is changes in methodologies. Software today is more and more designed using an iterative methodology where smaller chunks of code are delivered on a regular interval, including less new or modified features per delivery. The testing process has also changed to reflect this methodology. Instead of a longer test interval prior to the release of the product, smaller continuous test intervals are conducted. As code is continuously changed, retesting of previously delivered code is necessary. A need for regression testing of previously working code has been introduced.

Previous research in this focuses on reducing the scope of testing at each iteration when new functionality is delivered. The common way is to limit the test case volume by different means of code coverage metrics. Different examples of code coverage metrics includes statement/line, block, path, or function coverage to mention a few. They focus on limiting the volume of test cases.

Furthermore the iterative way of releasing new features in smaller chunks has increased the need for automation of the test process, where the research both covers tools and ways to create test cases. One of the common ways to create test case for a web applications is to use a record-replay method. Where the navigation of the code is manually created by recording how the tester navigates a web page. The recording is then replayed using various tools. This however increases maintenance to up keep the test cases when the navigation changes and the test case need to be recorded again.

Running the test cases in a GUI is considered both slow and difficult. Hence the approach has been to divide the testing focus. The GUI is tested from a look and feel perspective, rather than from a functional perspective.

Already when selecting a topic to research, the researcher has to ask him/herself what is the essential new thing that the research will add to already existing research papers and studies. The area of testing is already well researched. The research of the methodology, the selection and reduction in test case selection, the tools to use, the problematic areas arising when these are applied. As there is a lot of research already done, *so what is new with this research?*

Previous research has focused on methods to limit the scope of the testing in various ways. The problematic GUI has also been removed from the

equation due to the slowness and problems to keep test cases running, which increases the total volume of test cases needed for regression testing. The new research that this report includes is to research the problems from a new approach, instead of reducing the scope and removing components from the equation, it researches the possibility to combine test cases and how to address the issues with using the real GUI. Hence covering a larger scope of testing per used test case by:

1. Studying the problems that arise during the execution of the test case
2. The study is conducted on a commercially available web application
3. Measures to address issues that arise are also tested and evaluated during the study

The research is conducted on a commercially available web solution, which is not always the case when testing of web solutions is researched.

1.3 Problem formulation

Here you give a detailed description of the research problem you intend to investigate. You can re-use the problem formulation from your project plan. You can read more about research problems [here](#).

The activity of regression testing is testing that nothing has broken in the existing functionality of the application. However when the testing of an SPA is split into separate standalone activities the interoperability between frontend and backend components is not tested, or is only partly tested.

This introduces a risk that interoperability issues are never found during the regression test activity. The volume of test cases also increases, as several partly overlapping test cases are needed to cover each part as standalone components.

The navigation performed in the GUI requires its own set of test cases, with a simulated backend, and the business logic provided in the backend its own set, where the frontend is simulated in order to drive the test execution. Resulting in the partly overlapping test cases. The time that might be gained by regression testing the components separately comes with a cost in form of additional test cases and a risk that interoperability failures are never discovered during testing of the SPA

By studying and addressing the factors limiting the use of the GUI in regression testing of a commercially available web solution the intention is to show that the total amount of test cases needed can be reduced if the GUI is taken into use. The test coverage is also improved and fewer test cases are executed and maintained.

1.4 Motivation

Here you motivate why your research problem is interesting for science, society or industry. You can re-use the motivation part from your project plan.

Upto 80% of the testing budget can be spent on regression testing. As the usage of the real GUI in driving the testing is difficult it is vital to understand the factors causing these difficulties. By addressing them with proper measures the need to divide the testing activity into separate entities (frontend and backend) decreases. Reducing partly overlapping test cases and increasing the reliability in the test results, where interoperability is also tested.

1.5 Research Question

Your research problem is broken down into one or more research questions (RQ). RQs state exactly what you want to investigate in your thesis project. You have already defined research questions in your project plan. Copy and paste them here. You can read more about research questions [here](#).

RQ1	Research question 1...
RQ2	Research question 2...

You are also required to make statements about tentative and expected answers to your research questions (called propositions). What do you think your project will result in?

Don't mention anything about research method here. It will be covered in the next chapter.

RQ1.	What are the top 2 issues using Selenium WebDriver that cause longer test case execution times when test are run using a browser, compared to testing the same functionality with APIs, and what can be done to minimize the time differences?
RQ2.	What are the top 2 issues using Selenium WebDriver causing flaky (unreliable) test results, and how can these be overcome?
RQ3.	What are the top 2 issues using Selenium WebDriver causing test cases to break when GUI modifications are done, and how can these be overcome?

The expected outcome of the project is knowledge about the main issues preventing the Web GUI from being used as driver in regression testing of

the business requirements in a web based application. Thus preventing the whole application to be tested together.

The project also expects to propose measures to address the issues found and show the difference in the results when those measures are applied.

1.6 Scope/Limitation

You cannot solve everything. Here you describe what you do, and what you don't do, in your project. Limitations can for example be that you only compare some frameworks of all frameworks available on the market, that you only suggest an architecture for a specific software product and not a general architecture, or that you only include university students in a study and not a broader population sample.

There are many different test solutions available, providing different kind of functionality. This project limits the use of test tools to Selenium WebDriver for communication with the web browser.

The report doesn't compare different web browser driving tools. The aim with the report is to understand the underlying factors that motivates testing the two entities separately.

It is assumed that a lot of different factors come in play, however the project focuses on the factors that are related to execution time of the test case, robustness and reliability of the test result.

There are a lot of different SPAs available on the market and this project uses one of them, Svenskaspels' implementation of a few of their betting games. Some of the findings made might not be applicable to other SPAs, as this depends entirely on the nature of the issues that are found.

The test execution is for sure expected to be slower when tests are run via a web browser compared to using direct API calls for testing of the business logic. However a decent "good enough" execution time is expected to be achieved.

The SPAs uses JavaScripts to implement its functionality. This creates a dependency on the used Web browser, as they implement JavaScript in different ways, rendering different performances. The experimentation is limited to using Firefox 52.x.

1.7 Target group

Here you outline which target group that might be interested in your thesis. If you, for example, do a project about software architectures, a target group can be professional developers and architects that work with similar software systems as the system you investigated.

The target group is Software Testers of web applications that faces similar issues and any test organization that wishes to enhance the confidence of their regression testing.

1.8 Outline

Here you outline the rest of the report. It shall contain which chapters that will follow, and what each of them is about.

The following chapters outlines how the experimentation was done and what the result and final outcome of the experiment was.

Firstly the method is described to give an overview in how the project was conducted, and what data that was collected to address the stated questions. The method also describes how the proposed measures were applied and how they were tested to verify that they had a positive impact to the final result.

Secondly the implementation of the FIA application is described. This outlines how the project was run as well, as the tool was designed to collect the data stated in the method.

Thirdly the data collected by the FIA application is presented.

Followed by the final part of the thesis where the presented data is analyzed, discussed and summarized in a conclusion. The final part also addresses issues were further research would make sense.

2 Method

In this chapter you shall describe the scientific approach that will be used to answer your research problem.

2.1 Scientific Approach

Here you define which formal research method(s) that shall be used to answer your research question(s). Research methods are divided into quantitative and qualitative methods. Quantitative methods result in numerical data, and qualitative in non-numerical data. You can read about different research methods [here](#).

The stated research area contains aspects that need to be addressed using both a quantitative research method, and as well areas where a qualitative research approach is needed.

The quantitative research is needed to find the areas where most of the problems arise. Primarily used for understanding of *when* a problem occurs, and *what* are the problems that occur. Qualitative research is used to understand the *why* and *how* a problem occurs.

A combination of both methods are used in order to understand the full scope of *what*, *when*, *why* and *how*.

2.2 Method Description

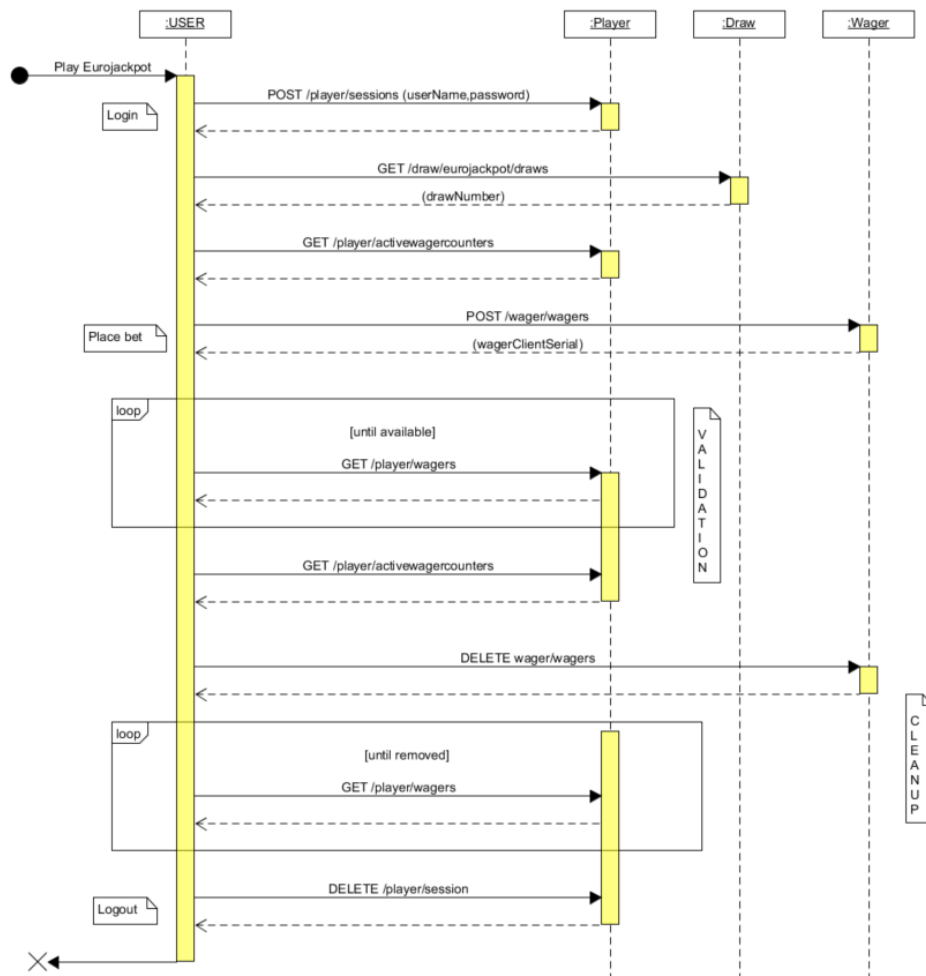
When you have defined which research method(s) that will be used to answer your research question(s), you shall continue by describing how the research methods will be applied in your project. If you for example shall conduct an experiment you describe your *independent* and *dependent* variables, how many times the experiment will be repeated, which software tools that will be used to run the experiment, etc. If you for example shall conduct a survey you describe how participants will be selected, how the questionnaire will be designed, how it shall be distributed to the participants, how data will be analyzed, etc.

The research approaches three main topics. Firstly the execution time required for the test execution. Secondly the correctness and reliability of the test result. Finally the area of ensuring that testing can continue even though changes are introduced in the GUI, which is used to drive the test execution.

Required execution time

The collection of the execution time spent to perform a task is a quantitative task. The execution time required for each API call is measured during the test execution. As the testing of a single business requirement involves several different API calls, the total time spent for testing a business

requirement is also collected. This results in both an overall picture of the total needed execution time, as well as a detailed picture of what API calls that are needed, and how much execution time each task consumes, see [figure 2.2.1] for an example of the API request flow to place a bet on one Eurojackpot row.



[Figure 2.2.1, API Flow Eurojackpot]

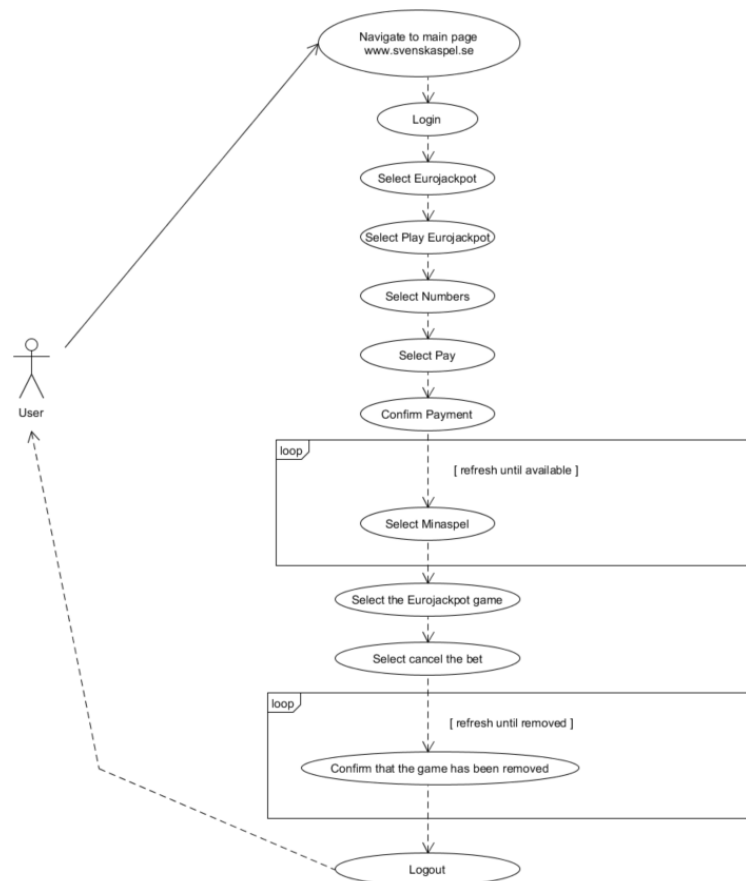
The same test case is then executed, this time using a GUI driven approach. In the GUI driven approach the execution times for each API calls can't be measured in the same way, as it is unknown what APIs the SPA actually is using. Instead the execution time of each task is measured, as well as the total required execution time, see [figure 2.2.2] for an example of the tasks required to place the same kind of bet, i.e. one Eurojackpot row using a GUI.

The results of the API data are compared with the results of the GUI driven testing. And the areas where the test execution times deviates the most is analyzed.

In advance it is not possible to say what kind of measures that might be needed in order to address the difference in execution times. However the following step is to modify the GUI navigation to optimize and improve the execution times. The modifications needs to be evaluated by retesting the same test case.

This is an iterative task which is assumed to be conducted several times before a good enough execution time has been achieved.

The delta in the implementation between the first test execution and the last, which can be considered as the required measures in order to improve the execution times. The addressed areas and the modified delta can be considered as answers to research question RQ1.



[Figure 2.2.2, GUI Flow Eurojackpot]

Reliability of test results

Tests driven using GUIs can often give flaky results. Sometimes they work and sometimes they don't. To be able to trust the received test result flakiness can't be tolerated.

Each of the used test sequences will be tested manually first to ensure that the business requirement is implemented and working. The used test suite is executed several times, with an expectation of a 100% success rate.

During the test execution any failure in the result is registered, as well as what task that failed. Each failure is then analyzed further to understand the reason behind the failure. Once the reason is known modifications to the navigation will be applied and the test case is retested, using the same approach as required for understanding and approving of the execution times. The addressed areas and the modified delta can be considered as answers to research question RQ2.

Modifications of the GUI

When the GUI is used to drive the testing, it introduces a vulnerability for breaking the test cases. New components could be introduced, or old ones removed or replaced.

The third research question addresses the issue of keeping the test cases running successfully. I.e. limit the effects caused by the vulnerability. Hence this question will be addressed once the previous two research questions have been answered. At that point it is expected that the test suit is executed fast enough and that the result is reliable. The third question is answered by analyzing how often the test execution fails and where the solution requires modifications to the navigation of the test execution. I.e. how often does one need to update the test case navigation and why.

Used Execution environment

It is not only the SPA that are continuously being changed, also the software components around the SPA are continuously being updated. Both the web browser and selenium, as well as drivers used are continuously being changed and improved. Old bugs are fixed, and new ones are introduced. The aim is to always use the latest software version of each test tool. This however requires a continuous retesting and reevaluation during the entire project. This reevaluation is taking place while investigation research question RQ3.

2.3 Reliability and Validity

Here you discuss the reliability and validity of your project. You can read about reliability [here](#) and about validity [here](#). Discuss if you have any reliability issues or validity threats in your project [here](#).

The software components used for the GUI navigation are continuously changed. Hence limitations, restrictions in the current implementation could affect the areas where most problems arise differently, depending on which version of the SW component that is used.

The experimentation is performed in four different test system. While the test systems provides similar functionalities, there are differences in the capabilities of the HW. Execution times of a test suite can vary depending on which test system that was used.

In some occasions the test system can also be shared with other users, which could have a negative effect on both the outcome of the test result, as well as the used execution time.

The SPA is using JavaScript to render its required functionality. This creates a dependency to the web browser that is used, as they implement JavaScript in different ways. Firefox versions 48-53 and Google Chrome 58.0 were used during the experimentation.

2.4 Ethical Considerations

You are required to discuss any ethical considerations (if any) in your project. If you do an experiment you will most likely not have any ethical considerations, but in a survey ethical considerations can for example be how you make sure that the privacy of the people participating in the study is not violated (by for example removing names from the gathered data).

The research is conducted in the form of experimenting on an existing product, no ethical considerations are foreseen.

3 Implementation

It is common that you will develop something in your project. It can be a mobile app, a stand-alone application, a website, a game, etc. In this chapter you describe the software you have implemented.

In some projects you don't develop anything, for example if you do a systematic literature review. In this case you remove this chapter.

In the project a number of different deliverables were implemented. Primarily a set of test cases to test the business requirement of placing the bet was implemented.

Secondly a test loader was implemented to execute the suite of test cases. The simple reason why a loader was implemented was simply to get full control of the execution without having to overcome with various tool related features or characteristics. Ordinarily a test tool comes with a set of generic components that generates a certain overhead in execution times, as well as needed workarounds to handle specific requirements. These were unwanted options that we removed by making an own implementation of a test loader.

In order to do this in a controlled way, with a flexible navigation that enables experimentation with the GUI navigation a test loader (FIA) was implemented [figure 3.1].

FIA is a platform independent JAVA implementation.

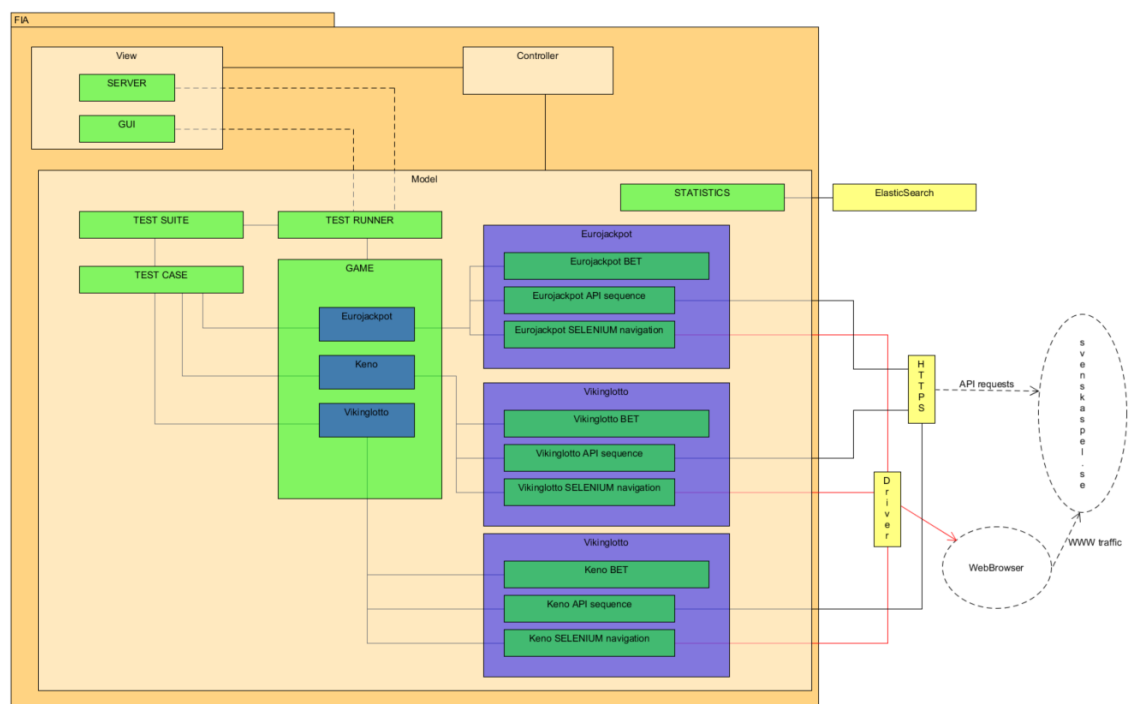


Figure 3.1 FIA overview

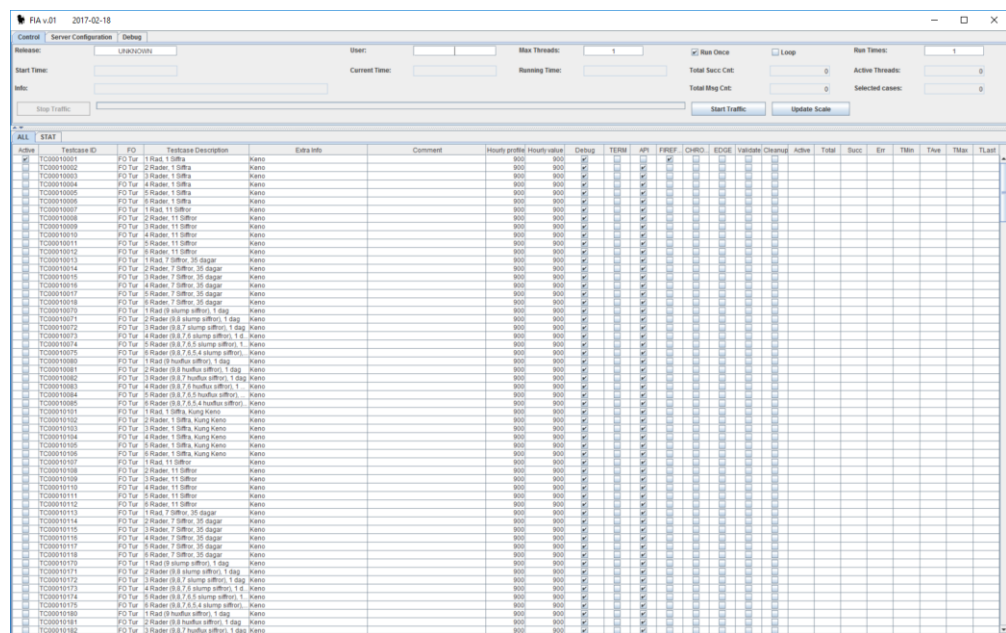
3.1 Components of FIA

View

FIA runs in two different modes, either with a GUI [figure 3.2] or in a server mode. The implementation of FIA is generic, and hence any web browser could be used for the execution. However Edge or Internet Explorer (IE) is only available on the Windows operating system.

Due to this there is only a very limited need for a pure server solution. As the browser depends on the desktop for the execution. The server mode is however used when running the API test cases, as those are text base, and doesn't depend on a desktop to execute.

The internal GUI [figure 3.2] of the tool is primarily for monitoring and follow up of the progress of the test execution.



The screenshot displays the FIA v01 GUI. At the top, there's a title bar 'FIA v01 2017-02-18'. Below it is a control panel with tabs for 'Control', 'Server Configuration', and 'Debug'. The 'Control' tab is active, showing fields for 'Release' (set to 'Unkown'), 'Start Time', 'Current Time', 'Max Threads' (set to 1), 'Run Once' (checked), 'Loop' (unchecked), 'Run Times' (set to 1), 'Total Sess Cnt', 'Active Threads', 'Total Mpg Cnt', and 'Selected cases'. There are buttons for 'Stop Traffic', 'Start Traffic', and 'Update Scale'. Below the control panel is a large table with columns: 'ALL', 'STAT', 'Active', 'Testcase ID', 'PO', 'Testcase Description', 'Extra Info', 'Comment', 'Hourly profile', 'Hourly value', 'Dropout', 'TERR', 'API', 'FREQ', 'CHRO', 'EDGE', 'Validate', 'Checksum', 'Active', 'Total', 'Succ', 'Err', 'Time', 'Time', 'Time', 'Time'. The table contains numerous rows of test case data, including IDs like TC00010001, TC00010002, etc., and descriptions like 'PO Tur 13 Rad. 1.5dgr', 'PO Tur 13 Rad. 1.5dgr', etc.

Figure 3.2 GUI of FIA

Model

The model is the Core implementation of FIA. It consists of a TestRunner, TestSuite, TestCases, supported Games and a Statistics component.

TestRunner

The TestRunner is a multi-threaded implementation. Upon start of execution of the test case, a new thread is created and the execution of the test case is contained in that thread. Upon completion of the test case the thread is killed. Hence each test

case is running independently of each other and thus one failing test case does not affect any of the other test cases [figure 3.3].

```
SingleTestCaseThread startTrafficCaseInstance = new SingleTestCaseThread (false);
startTrafficCaseInstance.setTestCaseId (statKey,(active+1),debug,at,apiUrl,webUrl,validate,cleanup);

Thread startTrafficThread = new Thread(startTrafficCaseInstance, ("FiaThread-" +statKey + "-" + (active+1)));
startTrafficThread.start();
```

Figure 3.3 Execution of Threads

The same is applicable if multiple instances of the same test case should be executed. At start of each case a unique thread for each test case is also started.

This approach was selected to avoid the loader becoming an issue related to RQ1-3. I.e. the loader should not cause the test cases to fail. Failure should be related to either the externally used SW components, the navigation or that testing of the business requirement(s) fails.

The test suite can be executed in different ways by the loader, where the following options are possible:

1. Case by Case (serial mode)
One test case at a time is executed. The following test case is started upon completion of the previous one.
2. Maximum number of Thread (parallel mode)
The test cases run in parallel mode upto the maximum number of allowed concurrent test cases.
3. Hourly Traffic profile (parallel mode)
A required amount of test threads are started per hour. The configuration is set per test case, not on the full test suite and the load is evenly spread out during the 1 hour time period.

In the first two test modes it is also possible to indicate how many times the entire test suite should be looped, or if it should run until manually stopped.

TestSuite

The test suit is a csv formatted list of required test cases, with a set of parameters, indicating the requested traffic level and interface to be used. Hourly profile indicates the required amount of test executions per hour. The execution is evenly spread out during the full hour. The interface to use indicates that the case should either be executed via the API or via the WEB interface.

TestCases

Each test case is implemented in its own class file, by extending the general test case class. The General class contains common functions such as setting ID, a common start up sequence, requesting a player user and a teardown sequence.

The logic of the test case is setup in the TestCase class [figure 3.4].

```

public class TestCase00030002 extends TestCaseGeneral{

    private final String TC_DESCRIPTION = "Eurojackpot(t2 Rader, 1 vecka";

    public TestCase00030002 (boolean debug, AccessType atype, String myUrl, boolean val, boolean clean){
        super (debug, atype, myUrl, val, clean);
        setTcIdAndDescription(this.getClass().getName(), TC_DESCRIPTION);
    }

    public void run(){

        // 1. STEP COUNTER OF ACTIVE THREADS
        stepThreadCounter ();

        // 2. FETCH DEFAULT DATA AND STEP COUNTERS
        if (getUser() == false) return;

        // 3. SETUP THE TEST CASE
        Eurojackpot eurojackpot = new Eurojackpot (url, username, password);
        eurojackpot.setFixedRad(debug, 14, 15, 16, 17, 18, 7, 8);
        eurojackpot.setFixedRad(debug, 15, 16, 17, 18, 19, 8, 9);
        eurojackpot.setBeta(1);

        Long startTime = System.currentTimeMillis();
        boolean result = eurojackpot.playEurojackpot(debug, at, validate, cleanup);
        Long stopTime = System.currentTimeMillis();

        // 4. END THE TEST CASE AND CLEANUP
        endTest (result, (stopTime-startTime));

    }
}

```

Figure 3.4 Test Case

Implementation of the Game components

Each game is implemented in its own set of core components [figure 3.5].

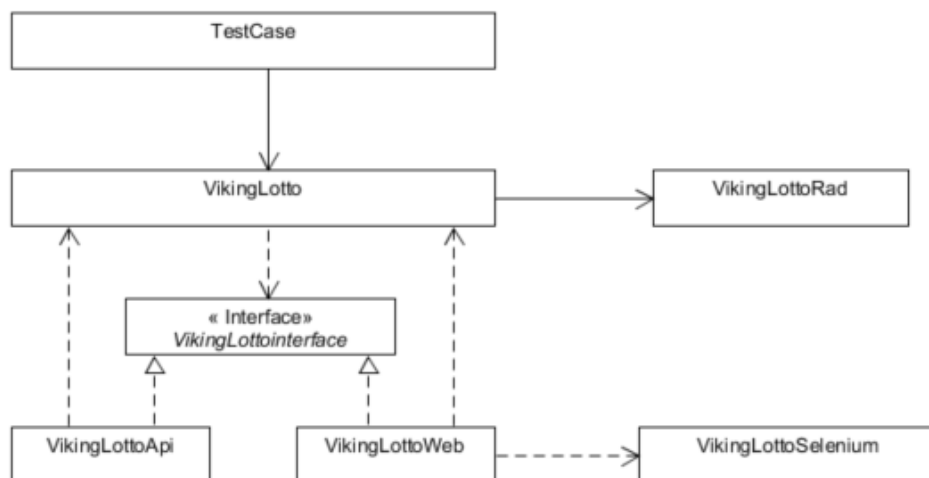


Figure 3.5 Game components

The required configuration on how to setup the bet is done in VikingLotto and VikingLottoRad. The sequence of API requests, or required navigation is implemented in VikingLottoApi respectively VikingLottoWeb, which also uses VikingLottoSelenium as a helper class, implementing the required navigation towards the Selenium package.

Each API message is implemented in its own class [figure 3.6], the only reason behind this decision is to make the implementation of each API as independent as possible [figure 3.7]. Also the data extractor is implemented in its own class for each API message. This results in partly duplicated code, which can be consider as bad practice from an object oriented point of view. However it reduces the risk of breaking other API requests in case something needs to be updated.

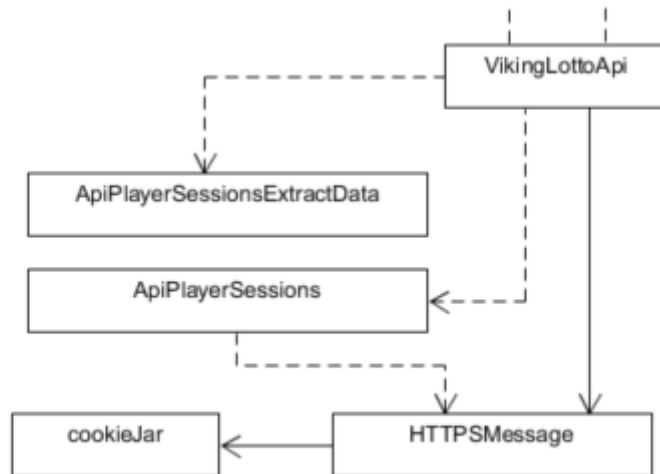


Figure 3.6 API Message sending

```

//-----
// LOGIN the USER
//-----

// CREATE a new HTTPS message
HTTPSMessage httpsLogin = new HTTPSMessage (HTTPSMessageType.POST,myCookie);

// ADD the message to the list of sent messages
messages.add(httpsLogin);

// SEND the MESSAGE
ApiPlayerSessions apiSession = new ApiPlayerSessions (apiUrl);
boolean result = apiSession.postPlayerSessions(httpsLogin,userName,userPassword);
result = validateResponse (httpsLogin, result, debug, ("LOGIN PLAYER: "+userName +"\t" +userPassword ));

if (debug == true) System.out.print("\nDEBUG : " +httpsLogin.getResponseMessage());
if (result == false){
    EsInterface.getInstance().LogErrorEvent("API","/player/sessions","VikingLotto", 0L, "", ("Failed to Login"+userName));
    return false;
}
  
```

Figure 3.7 API Message sending example

The web interfaces uses the selenium WebDriver packaged to drive the navigation of a web browser. In this case the VikingLottoWeb is the core component for executing the sequence when a bet is to be placed [figure 3.8]. VikingLottoSelenium can be seen as a helper class to VikingLottoWeb, it implements the more detailed navigation components of the web page, such as how to click a button, selecting numbers etc.

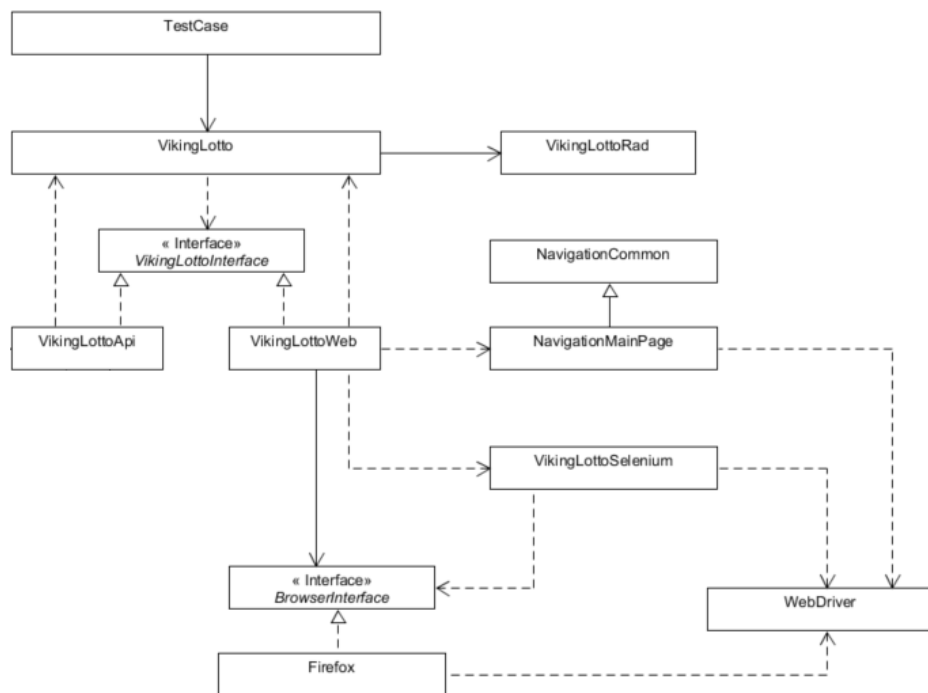


Figure 3.8 Web navigation

This architecture enables easy maintenance of the navigation. By using high cohesion the following is achievable in this architecture:

1. If a new type of web browser is introduced the component affected is mainly the Firefox component which is replaced by the unique driver of the new web browser [figure 3.9].
2. The main page navigation is common for all web browsers and all games
3. A new or different game affects only VikingLottoWeb (handling the sequence flow of the web page) and VikingLottoSelenium (implementing the Vikinglotto specific selenium navigation)

The main difference between the different games is the navigation sequence required to place the bet of the game. The implementation of the game Keno uses the same structure as the Vikinglotto game [figure 3.10]. The core components which are specific for the game are the only ones replaced.

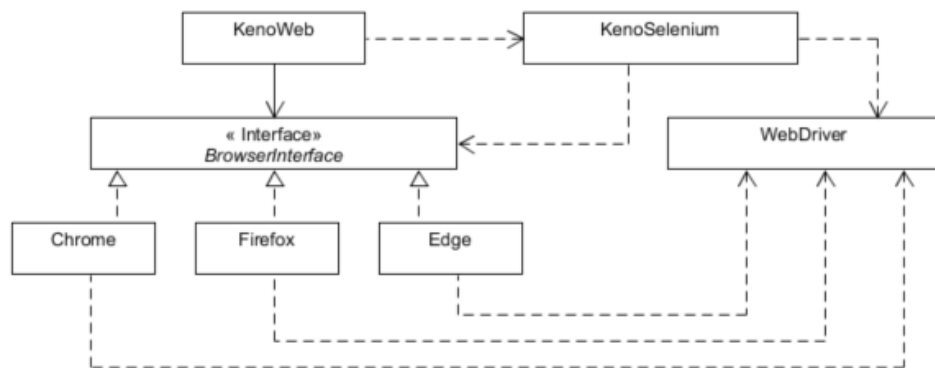


Figure 3.9 Web browsers

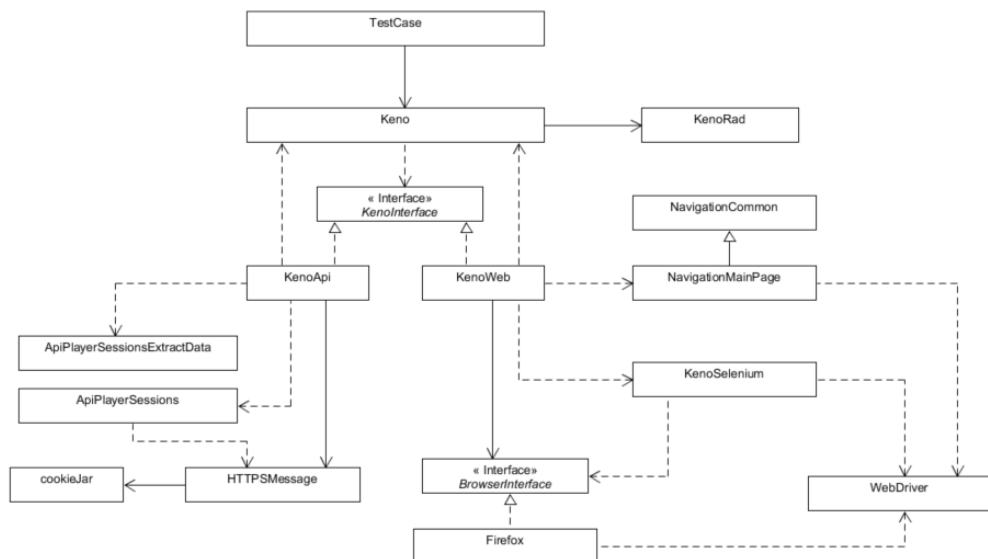


Figure 3.10 Keno Overview

Statistics

Implements a singleton providing an interface for storing data in an ElasticSearch database. It also holds the statistical data presented in the internal GUI when the tool is running in the GUI mode.

4 Results

In this chapter you show and describe your results. You shall only show the raw results without any analysis, and you shall not put any conclusions or opinions in the description of the results. Try to be as objective as possible. An example of results from an experiment comparing five sorting algorithms is shown in Table 4.1 below.

Run	Bubble	Quick	Selection	Insertion	Merge
1	17384	24	3258	3	30
2	17559	21	3386	3	27
3	17795	19	3344	4	28
4	17484	20	3417	3	28
5	17642	19	3358	3	30
Average	17572.8	20.6	3352.6	3.2	28.6

Table 4.1: Execution times for the five sorting algorithms on 100 000 random numbers between 0 and 10 000.

What you show heavily depends on the type of research and what type of data you collect. Numerical data can for example be shown in both tables and graphs. A complementary graph for the sorting algorithms example is shown in Figure 4.1. For a questionnaire you can show the frequency (how many participants that selected the same answer) of each possible answer to a question.

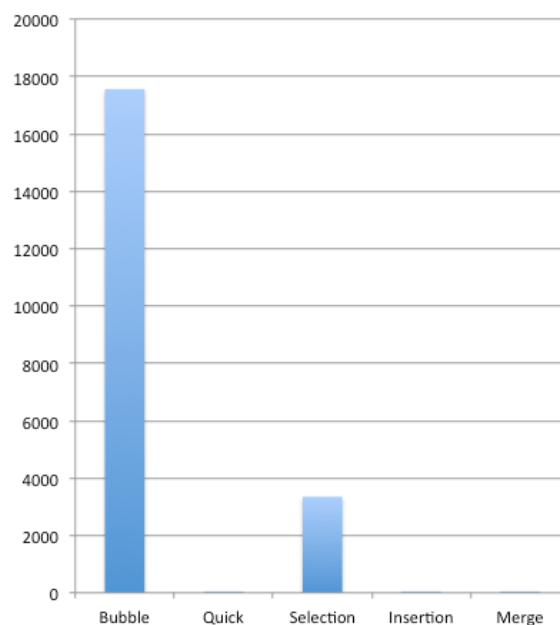


Figure 4.1: Execution times for the five sorting algorithms shown as a graph.

Note that Tables and Figures shall be labeled with chapter.number, for example Table 4.1 and Figure 1.6.

4.1 Test case selection

The selected test cases are a subset of the total test suite. The cases selected consists of cases that requires only a few elements to be selected as well as cases where more elements have to be selected in the GUI. Ranging from the simplest test case with the bare minimum of selectable elements to the worst test case with the maximum of elements to be selected.

<Should this be moved to the appendix instead??>

Game	Test Case	Description
Keno	TC00010001	1 Row, 1 Digit
Keno	TC00010006	6 Rows, 1 Digit
Keno	TC00010007	1 Row, 11 Digits
Keno	TC00010012	6 Rows, 11 Digits
Keno	TC00010500	Package game 30
Keno	TC00010503	Package game 250
Keno	TC00010600	System game 2, 3 digits
Keno	TC00010670	System game 7, 8 random digits

Table 4.1.1: Selected Test cases Keno

Game	Test Case	Description
Vikingslotto	TC00020001	1 Row
Vikingslotto	TC00020012	12 Rows
Vikingslotto	TC00020402	Package game 60
Vikingslotto	TC00020412	Package game 120
Vikingslotto	TC00020502	System game 6 + 2
Vikingslotto	TC00020508	System game 6 + 8
Vikingslotto	TC00020538	System game 9 + 8
Vikingslotto	TC00020544	System game 10 + 4
Vikingslotto	TC00020551	System game 11 + 1
Vikingslotto	TC00020561	System game 12 + 1

Table 4.1.2: Selected Test cases Vikingslotto

Game	Test Case	Description
Eurojackpot	TC00030001	1 Row
Eurojackpot	TC00030002	2 Rows
Eurojackpot	TC00030008	8 Rows
Eurojackpot	TC00030041	1 Random row
Eurojackpot	TC00030048	8 Random rows
Eurojackpot	TC00030071	1 Row Huxflux
Eurojackpot	TC00030078	8 Rows Huxflux
Eurojackpot	TC00030100	Package game 200
Eurojackpot	TC00030110	Package game 200 Huxflux
Eurojackpot	TC00030200	1-8 Random rows
Eurojackpot	TC00030210	1-8 Huxflux rows
Eurojackpot	TC00030318	System game 6 + 10
Eurojackpot	TC00030325	System game 7 + 7
Eurojackpot	TC00030370	System game 12 +2

Table 4.1.3: Selected Test cases Eurojackpot

For a comparable analysis between the usage of API calls and GUI navigation using a web browser, the scenarios are divided into 8 steps.

Step	Description	Comment
1	Start a web browser	Not applicable when API calls are used
2	User Login	
3	Initialize the game	
4	Place the bet	
5	Validate placed bet	
6	Remove bet and validate bet removed	
7	User Logout	

Table 4.1.4: Scenario steps

4.2 Test case execution times

Test case	API Execution Time (ms)	Google Chrome Execution Time (ms)
TC00010001	2030,33	21176,29
TC00010006	2043,56	32122,34
TC00010007	2012,89	22142,45
TC00010012	2032,14	42123,32
TC00010500	2014,99	19344,95
TC00010503	2038,22	21201,12
TC00010600	2044,08	19635,88
TC00010670	1998,57	23124,21
TC00020001	2007,95	21229,98
TC00020012	2013,37	43713,64
TC00020402	2055,22	19212,04
TC00020412	2000,32	19476,82
TC00020502	2001,32	19863,34
TC00020508	2001,13	20362,11
TC00020538	2018,53	22121,11
TC00020544	2028,40	23123,43
TC00020551	1986,25	21342,23
TC00020561	2044,53	19921,21
TC00030001	2021,68	24638,27
TC00030002	1998,88	31164,10
TC00030008	2035,48	72915,60
TC00030041	2023,46	24836,10
TC00030048	2002,03	71885,00
TC00030071	2033,42	20714,20
TC00030078	2019,82	75231,12
TC00030100	1954,78	22131,21
TC00030110	2040,30	30123,12
TC00030200	2020,78	51213,21
TC00030210	2016,99	68231,21
TC00030318	2013,54	25612,32
TC00030325	1997,28	23123,32
TC00030370	2029,58	27213,12

Table 4.2.1: Average execution times of the test case from 100 test runs

<Dubbel kolla alla värden!!! Testa mot den nya Google Chrome drivern samt senast SW versionen I-T3, fixa en backup på excel filen>

Game	Interface	Step 1 (ms)	Step 2 (ms)	Step 3 (ms)	Step 4 (ms)
Keno	Api	*	36,96	54,85	29,52
Keno	Google Chrome	3340,67	2736,26	1518,63	3682,89
Vikinglotto	Api	*	36,26	71,83	29,32
Vikinglotto	Google Chrome	3366,79	2763,19	1592,26	2624,54
Eurojackpot	Api	*	35,95	71,54	49,45
Eurojackpot	Google Chrome	3346,82	2729,21	**check this 1601,22	21213,23 ** check this

Table 4.2.2: Average execution times for scenario steps 1-4

* Not applicable

Game	Interface	Step 5 (ms)	Step 6 (ms)	Step 7 (ms)
Keno	Api	948,64	**	27,32
Keno	Google Chrome	1865,36	3480,07	223,63
Vikinglotto	Api	937,06	**	32,12
Vikinglotto	Google Chrome	2455,27	3385,41	229,85
Eurojackpot	Api	973,91	**	28,21
Eurojackpot	Google Chrome	2892,12	4120,12	341,12

Table 4.2.3: Average execution times for scenario steps 5-8

* Not applicable

** Included in the figures for Step5 ()

<Dubbel kolla alla värden något verkar lite skumt efter körning 84!!!>

<Kolla om man kan låna I-T3 en dag och köra om API biten för att få samma refrence Step5 + Step6 mellan API och WEB>

<Extract the data per test case in the GUI for a few examples showing 1 row vs many rows etc. This data is currently missing but shows better how time is being spent, add the same for Eurojackpot spinners>

<Table 4.4 Separated execution times 1 row, many rows>

4.3 Flaky test results

The result of the test cases were grouped into 4 categories. Successful, Hanging browser session, Navigational errors, and functional errors.

A hanging browser connection was detected by means of a timeout when the browser did not respond to any of the selenium calls for 20 seconds.

Navigational errors were detected if the browser was responsive but the element could not be located or accessed for a repeated number (100) times.

Functional errors were detected when the navigation was successful but for some reason the functionality was not provisioned by the system.

4.3.1 Test results over time for Vikinglotto

Date	Success Rate (%)	Hanging browser session (%)	Navigational errors (%)	Functional errors (%)
2016 w46	10	40	30	20
2016 w47	60	20		20
2016 w48	54	26		20
2016 w49	32	28	10	30
2016 w50	18	62		20
2016 w51	47	33		20
2016 w52	70	30		
2017 w01	64	36		
2017 w02 *	80		20	
2017 w03	80		20	
2017 w04	80		20	
2017 w05	80			20
2017 w06	80			20
2017 w07				100
2017 w08	100			
2017 w09	100			
2017 w10	100			
2017 w11				100
2017 w12				100
2017 w13				100
2017 w14	100			
2017 w15				100
2017 w16				100
2017 w17	100			

Table 4.3.1: Average execution result over time for Vikinglotto

*) In week 2, 2017 the browser used was switched from Mozilla Firefox to Google Chrome.

4.3.2 Test results over time for Keno

Date	Success Rate (%)	Hanging browser session (%)	Navigational errors (%)	Functional errors (%)
2016 w47	60	25	15	
2016 w48	64	18	18	
2016 w49	72	28		
2016 w50	85	15		
2016 w51	60	10	30	
2016 w52	74	16	10	
2017 w01	82	18		
2017 w02 *	100			
2017 w03	100			
2017 w04	100			
2017 w05	100			
2017 w06	100			
2017 w07	100			
2017 w08	100			
2017 w09	100			
2017 w10	100			
2017 w11	100			
2017 w12	100			
2017 w13	100			
2017 w14	100			
2017 w15	100			
2017 w16	100			
2017 w17	100			

Table 4.3.2: Average execution result over time for Keno

*) In week 2, 2017 the browser used was switched from Mozilla Firefox to Google Chrome.

4.3.3 Test results over time for Eurojackpot

Date	Success Rate (%)	Hanging browser session (%)	Navigational errors (%)	Functional errors (%)
2017 w04	71			29
2017 w05	71			29
2017 w06	71			29
2017 w07	71			29
2017 w08	71			29
2017 w09	79			21
2017 w10	79			21
2017 w11	79			21
2017 w12	100			
2017 w13	100			
2017 w14	100			
2017 w15	100			
2017 w16	100			
2017 w17	100			

Table 4.3.3: Average execution result over time for Eurojackpot

The functional errors in Eurojackpot was due to a faulty of navigation implementation in the test tool.

4.4 Modifications to GUI navigation

The classification for a needed modification of the GUI navigation was that an update to the tool was necessary to locate an element in the GUI, when it previously had worked.

Keno and Eurojackpot were not modified during the time period, however they share common elements with the other games, which were modified during the period. Vikinglotto was one of the games that was designed during the time period.

Date	Keno	Vikinglotto	Eurojackpot
2016 w46		7	
2016 w47			
2016 w48			
2016 w49		1	
2016 w50			
2016 w51			
2016 w52			
2017 w01			

2017 w02	3
2017 w03	1
2017 w04	3
2017 w05	
2017 w06	
2017 w07	
2017 w08	
2017 w09	
2017 w10	
2017 w11	
2017 w12	
2017 w13	
2017 w14	
2017 w15	
2017 w16	
2017 w17	

Table 4.4.1: Number of times GUI navigation was updated

5 Analysis

Here you give meaning to and your own opinions of the results. What conclusions can you draw from the results? It is important that you don't draw any conclusions that cannot be backed up by your data. Consider using statistical tests to back up your claims. You can read about statistical testing [here](#).

5.1 Required execution times

The API sequence is very similar regardless of which game that is played and the number of rows that are being placed in the bet. Hence it is anticipated that the variation in execution time being spent on placing the bet is very small. This can be seen in table 4.2.1 where the test execution times are very close to 2 seconds per test case regardless of which bet that is placed.

The minimum flow required to place a bet, without any validations, are 3 API calls. The API calls are login, read draw number, and place the wager. Once the bet has been placed there is a delay in the system until the component responsible for lookup of information on placed bets can access the data. Hence it is anticipated that most of the execution time spend in the API sequence flow is consumed in the validation part. I.e. validating that the bet has been placed and validating that the bet has been removed, this is seen in table 4.2.2 and 4.2.3.

The GUI navigation on the other hand is very similar in the initial parts, i.e. startup and navigation to the page where the bet can be placed, as well as the validation and removal of a placed bet. The main time is being spent on actually placing the bet where the execution time varies based on how many rows are being played, this can be seen in table 4.2.2, 4.2.3 and 4.2.4.

The additional execution time in placing the bet are the different effects implemented to give the player a better user experience, something that is not needed in pure API case.

The trade of for speed comes with an improved user experience with additional information that the game has been placed, confirmation popups to ensure that a specific task should be executed and that a task has been completed. The difference for placing a single row game is thus approximately an increased execution time of 10 times when the browser is used to drive the test compared to executing the test case using the API calls. Increasing the execution times from roughly 2 seconds to 20 seconds.

5.2 Flaky test results

The selection of tooling has a major impact on the test result. Mozilla Firefox and the GeckoDriver are still being developed and as can be seen in table 4.3.1, 4.3.2 and 4.3.3 the successful test execution results is improved after

the browser is changed 2017 week 2. The same code driving the test case was used, hence the only modification was the new interface towards Chrome and its driver. Therefore unquestionably the browser replacement was a major contributor to the increased success rate.

When the instability of the browser was removed the real problems causing flakiness in the test results were more visible, and listed as navigational errors in table 4.3.1, 4.3.2 and 4.3.3. Due to the difficulty to automatically analyze the cause, a screen dump was made when the problem occurred and the screen dump was manually analyzed. It shows two main areas of concern. Either the element can be found in the DOM, but for some reason it is not accessible or secondly the element is not found in the DOM, indicating that the sequence flow of the navigation is not as expected.

Vikingslotto is a new game and a number of times the whole test suite failed. Mainly due to other testing being conducted in the same test environment. This is shown in table 4.3.1 as a 100% error rate for the function. The test case failed as it was not possible to place the game, as the functionality was temporarily disabled.

The Eurojackpot test cases listed as functional failures were missing implementation in the test tool.

5.3 Keeping test cases running

The API sequence was not updated at any time during the data collection.

The GUI navigation was only updated for Vikingslotto at a few occasions. In total 15 occasions broke the test case due to some GUI element being replaced, seen in table 4.4.1.

Eurojackpot and Keno were both existing games and not redesigned during the time interval when the data was collected, only Vikingslotto was a new product, hence even though a number of GUI elements are common for all the existing games the navigation only broke on a few occasions.

6 Discussion

Here you discuss your findings and if your research question(s) have been answered. Think of research as a feedback loop. You define a problem, find a method of approaching it, execute the research and gather data. The data is then used to answer your research problem, thus creating the loop.

You shall also discuss how your findings relate to what other researchers have done in the area. Are your results similar to the findings in the related research you described in the Previous Research section?

This chapter is typically written in the present tense, while the previous chapters typically are written in past tense.

6.1 Test execution times

<To be added>

<Lista datat från spinners och olika GUI bitar>

6.2 Flaky test results

In the beginning Mozilla Firefox was the only web browser used, however due to the flaky results and continuously hangings of the GUI navigation a second browser Google Chrome was also taken into use. The difference can be seen in an increase of the success rate as well as the fact that hanging browser sessions were no longer detected. The browser was swapped starting from 2017 week 2, see tables 4.3.1, 4.3.2 and 4.3.3.

The remaining navigational errors that sometimes cause a test case to fail was analyzed by looking at the dumped screen shots when the fault occurred.

Firstly the reasons behind most of the failures were related to an element being in a state where it could not be selected, as it was covered by another element see fig 6.2.1, 6.2.2, and 6.2.3 for examples.



Figure 6.2.1 Digits 1-8 hidden under the banner

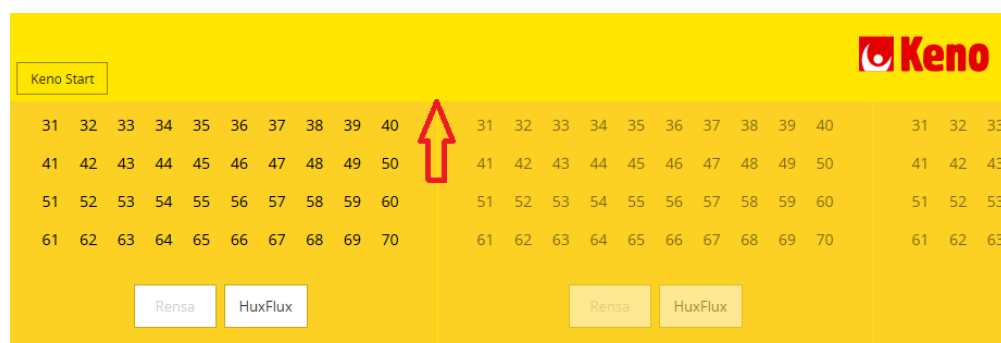


Figure 6.2.2 Digits 1-30 hidden under the banner

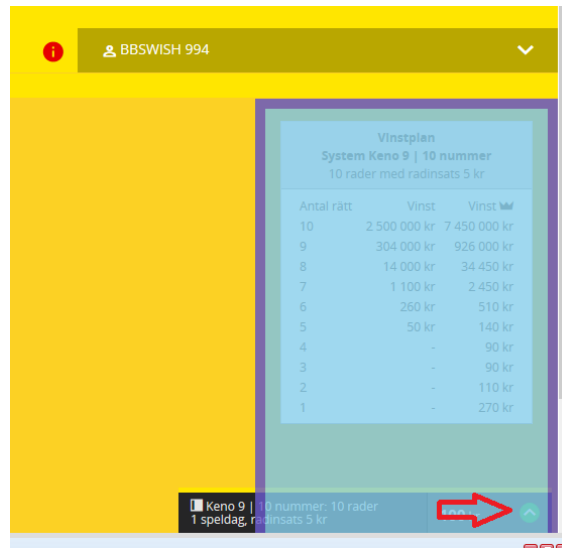


Figure 6.2.3 Button hidden under the info window

Even though the element is hidden it is still visible in the DOM, which selenium is interacting with.

The solution applied was to ensure that the element was always fully visible prior to selecting it. The component was scrolled into view before being clicked, see figure 6.2.4, 6.2.5 and 6.2.6 for a solution.

```
public boolean playRow (int ruta, ArrayList <Integer> numbers){
    // Align the Row to play
    if (centerRow(ruta) == false){
        return false;
    }

    if (numbers != null){
        for (int loopMe=0;numbers.size()>loopMe;loopMe++){
            boolean result = setKenoNumber (ruta,numbers.get(loopMe));
            if (result == false) return false;
        }
    }

    return true;
}
```

Figure 6.2.4 playRow method

```
// .//*[@id='boardsRegion']/div/div[1]/div
public boolean centerRow (int ruta){
    By kupongRuta = By.xpath(".*//*[@id='boardsRegion']/div/div["+ruta+"]/div");

    if (locateElement(kupongRuta)==false) return false;
    WebElement currentElement = findElement (kupongRuta);
    if (currentElement == null) return false;
    clickElement(currentElement);

    return true;
}
```

Figure 6.2.5 centerRow method

```

protected boolean locateElement (By currentXpath){
    for (int retry=0;MAX_RETRY>retry;retry++){
        try {
            WebElement currentElement = findElement (currentXpath);
            if (currentElement != null){
                JavascriptExecutor je = (JavascriptExecutor) driver;
                je.executeScript("arguments[0].scrollIntoView(true);",currentElement);
                return true;
            }
        }
        catch (StaleElementReferenceException e){
        }
        catch (TimeoutException e){
        }
    }
    return false;
}

```

Figure 6.2.6 locateElement method

Secondly there were unexpected popups that was not part of the navigational flow that is considered to be the normal flow of placing the bet, see figure 6.2.7-6.2.11 for examples.

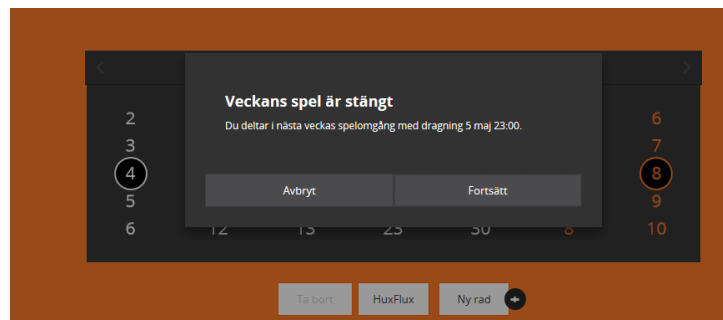


Figure 6.2.7 Popup closed for new bets

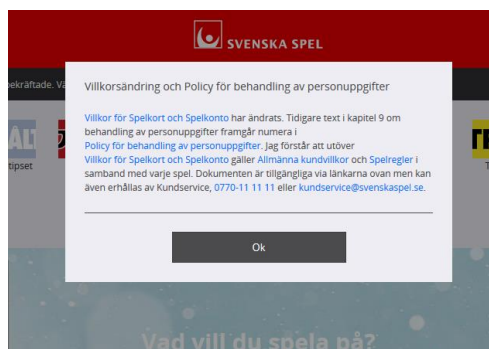


Figure 6.2.8 PUL requirements changed

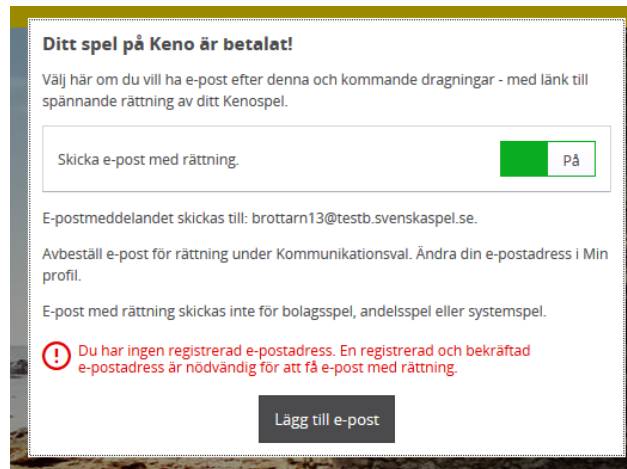


Figure 6.2.9 Result mail popup

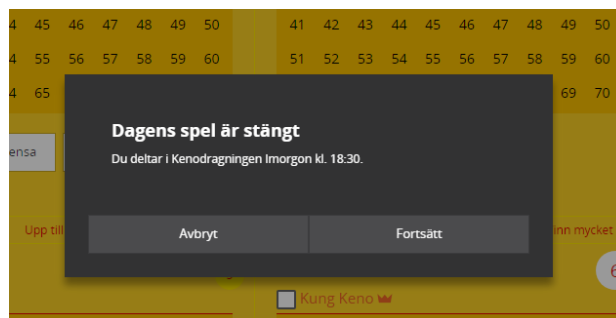


Figure 6.2.10 Keno game closed

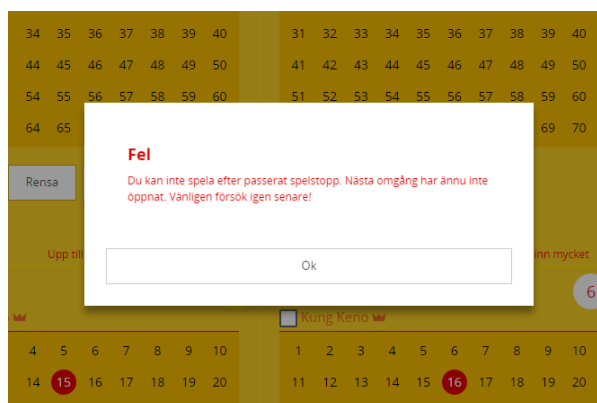


Figure 6.2.11 Keno game closed while placing bet

Two measures were applied to limit the effects of unconsidered popups. Prior to starting the test run all users are validated and all flags related to popups are cleared. The timing of when a test run is executed is also implemented to avoid games closed popups.

The main issues causing flaky test results were elements being hidden and unexpected popups. After applying corrective actions to these two issues test cases are executed successfully with a 100% success rate, thus answering RQ2.

6.3 GUI navigation requiring modifications

Vikingslotto was the only game being heavily modified during the time period. This can be considered as a reason why the required number of changes, see table 4.4.1 is low. However the implementation of the test tool already considered the probability of changes by locating objects on a page using the names displayed rather than a fixed path, see figure 6.3.1, 6.3.2 and 6.3.3 for an example, as the probability of name changes is less than components being moved around.

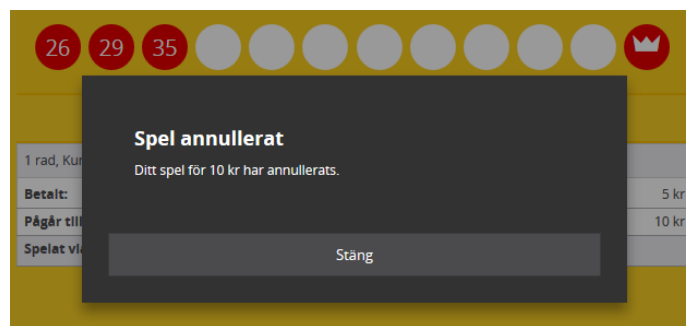


Figure 6.3.1 Close button

```
public boolean selectAnnulleraClose(){  
    return closePopUp ("Stäng");  
}
```

Figure 6.3.2 selectAnnulleraClose method

```

private boolean closePopUp (String buttonText){

    final int MAX_RETRY = 600;

    for (int loopme = 0;MAX_RETRY>loopme;loopme++){

        By closeButton = By.xpath("//*[@type='button']");
        List <WebElement> closeObject = findElementsList (closeButton);

        try {

            for (int loopme2=0;closeObject.size()>loopme2;loopme2++){

                WebElement closePopUp = closeObject.get(loopme2);

                if (closePopUp.getText().compareToIgnoreCase(buttonText)==0){
                    if (locateElement (closePopUp) == false) return false;
                    clickElement(closePopUp);
                    return true;
                }
            }

        }
        catch (StaleElementReferenceException e){

        }

    }

    return false;
}

```

Figure 6.3.3 closePopup method

If this contributed to the low number of changes or not can't be determined. Thus even though RQ3 is addressed, the design decision to use the name of objects to locate their elements shows a good result, but it lacks a reference implementation to verify the design towards. RQ3 is addressed, but the result can't be fully verified. Hence even though the results are positive, a proper answer to RQ3 can't be verified with the made research.

7 Conclusion

In this chapter you end your report with a conclusion of your findings. What have you shown in your project? Are your results relevant for science, industry or society? How general are your results (i.e. can they be applied to other areas/problems as well)? Also discuss if anything in your project could have been done differently to possibly get better results.

This chapter is also written in present tense.

7.1 Future Research

You cannot do everything within the limited scope of a thesis project. Here you discuss what you would do if you had continued working on your research project. Are there any open questions that you discovered during the project work that you didn't have time to investigate?

Here you shall include a list of all references used in your report. The reference list shall use the IEEE format. You can read about IEEE referencing [here](#). In the reference list below you can find examples of how to list a webpage [1][2], a journal article [3], a book [4] and a conference proceeding (article) [5].

References

[1] Linnaeus University. (2015) Course Room for Degree Projects. [Online]. Available: <https://mymoodle.lnu.se/course/view.php?id=5297#section-4>

[2] Monash University. (2015, Oct. 13) Citing and referencing: IEEE. [Online]. Available: <http://guides.lib.monash.edu/citing-referencing/ieee> ^[1]_{SEP}

[3] C. Lynch, “Big data: How do your data grow?” *Nature*, vol. 455, pp. 28–29, 2008.

^[1]_{SEP}

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

^[1]_{SEP}

[5] D. Agrawal, S. Das, and A. E. Abbadi, “Big data and cloud computing: current state and future opportunities,” in *Proceedings of the 14th International Conference on Extending Database Technology*, 2011, pp. 530–533.

[a1] Monash University () Yuan-FangLi n, ParamjitK.Das,DavidL.Dowe “Two decades of Webapplication testing — A survey of recentadvances”

[page object pattern]

<https://martinfowler.com/bliki/PageObject.html>



A Appendix 1

In the appendix you can put details that does not fit into the main report.
Examples are source code, long tables with raw data and questionnaires.