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

Digitalization has changed our world and how we interact with different systems, desktop applications have more and more been integrated with the internet. Where the web browser has become the GUI in today's system solutions, a change that needs to be considered in the automated regression testing process. Using the actual GUI has over time shown to be a complicated task and is therefore often broken out as its own standalone test object. This study looked into constrains of using the GUI as driver of the regression testing of business requirements in a web based solution, by evaluating the differences in execution times of test cases between API calls and GUI driven testing, flakiness of test results and required modifications over time for a specific test suite. Constraints were analyzed by looking into how reliability of the test results could be achieved. With a GUI driven full end to end scope the quality in software solutions could be improved by reducing the number of interface issues and detected errors in deployed systems. It would also reduce the volume of test cases that needs to be executed and maintained as there are no longer standalone parts to verify separately with partial overlapping test cases. The implementation utilized Selenium WebDriver to drive the GUI and the results showed that by utilizing Selenium the test execution times were increased from approximately 2 seconds (API) to 20-75 seconds (Selenium). The flaky test results could be eliminated by applying the appropriate pattern to detect, locate, and scroll into visibility prior to interacting with the element. In the end of the study the test execution result was 100% reliable. The navigation required 15 modifications over time to keep running. By applying the appropriate pattern a reliable test result can be achieved in end to end regression testing where the test case is driven from the GUI, however with a significant increase in execution time.

Keywords: Selenium, WebDriver, Regression testing, GUI driven testing

Preface

This thesis would not have been possible without the help of all marvelous colleagues at Svenska Spel AB. I’m grateful for all input, comments and to all those who acted as soundboards during the time I spent implementing FIA and writing this thesis.

A special thanks to Michael Olofsson and Jonas Wadsten for letting me share your test resources during the testing of the Vikinglotto game.

To the ladies in my life Arja, Emilia, Hilma and Fia you are my inspiration and the once that makes everyday a Joy to live, you are all dear to me and I can’t express enough how much I love you all.

Contents

1 Introduction

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) [1], [2] 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

An SPA is logically divided into two different entities, see figure 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.

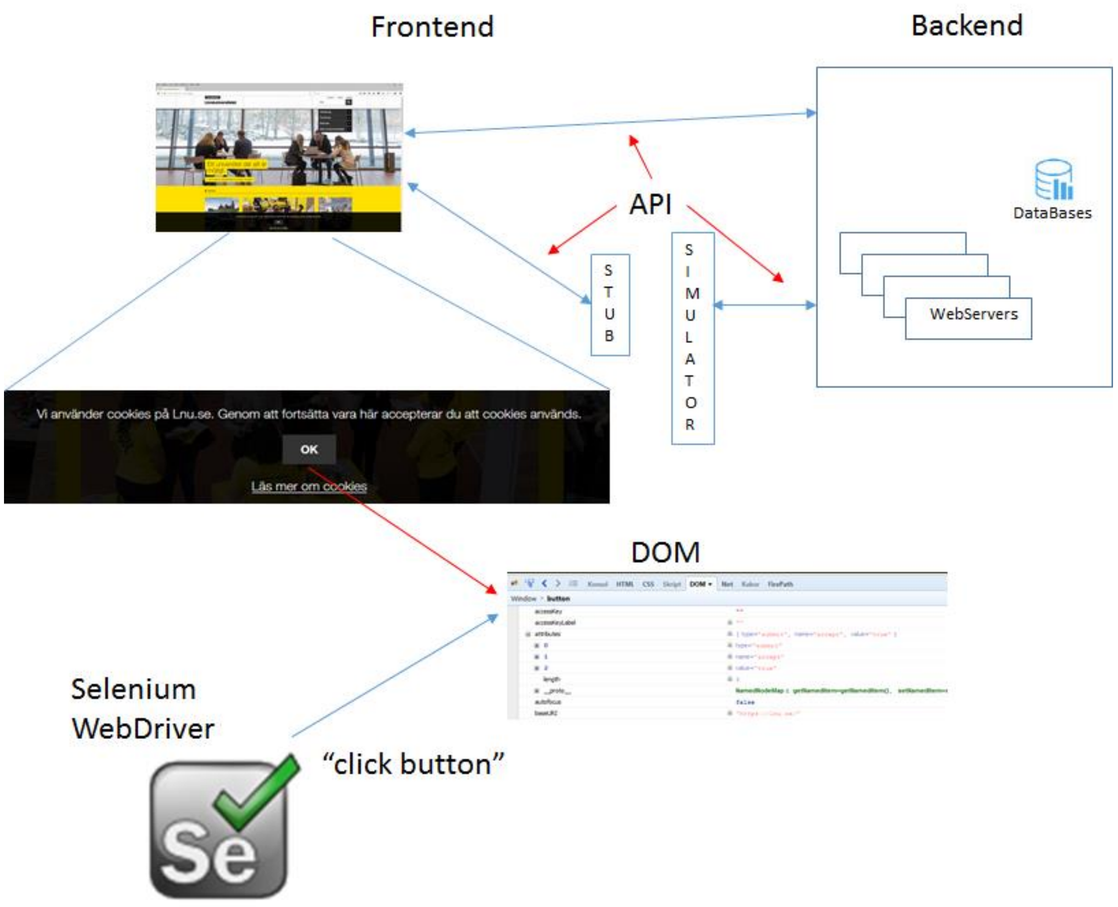


Figure 1.1 Test Automation of SPA

1.1.1 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 [3] 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 it is one of the most expensive test activities, indicated by Chittimalli et al. [4] 80% or more of the testing budget is consumed by regression testing. A further 50% of the total budget for software maintenance is spent on testing.

The IEEE definition of regression testing [5] 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.*

1.1.2 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.

1.1.3 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.

1.1.4 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.

1.1.5 API

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

1.1.6 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.

1.1.7 Document Object Model

The web browser keeps track of the web elements implemented by the SPA in an internal register, document object model (DOM) [6], [7]. 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.

1.1.8 Selenium WebDriver

Selenium WebDriver [8] 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 strings of text 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.

1.1.9 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

As indicated by Zarrad A. [9] regression testing as such is a research area that is covered. The main focus has been on more traditional testing techniques such as test cases select and test case reductions, regression testing of both GUIs and Databases. The Systematic review performed by Zarrad however indicates that regression testing of web based applications are not commonly researched.

Automation of regression testing can be conducted using tools such as Jmeter [10] or other Capture and Replay tools as indicated by Leotta M., et al. [11] or by applying tools which are programmatically in control of the browser driving the tests [12], [13]. The difference is both in how the test case is designed as well as how it is executed. In a Capture and Replay scenario the interaction with the web browser is recorded while performing a specific task. The recording is then later on replayed during testing. This however makes the test case vulnerable for changes and would break the test execution as indicated by Leotta M., et al [13]. Using a Capture and replay approach regularly broke the test execution and rerecording of test cases on a regular basis were required [14].

Other research in the area of GUI or GUI driven testing has focused on the page object pattern utilized in order to limit the need for maintaining test cases [14], [15], [16]. Where objects are logically group instead of treated as a single objects. Typically the logging box is considered as a page object of login requiring a username, password and clicking of the login button. Usage of the page object pattern has shown an increase in productivity with a reduction in cost of implementation

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

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

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

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

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, Svenska Spels' implementation of a few of their betting products. 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 Google Chrome 58.x.

1.7 Target group

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

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

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.

2.1 Scientific Approach

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

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.

The independent variable is the set of test cases used (tables 4.1-4.3) while the dependent variables are the once affected by the test run, namely execution time, test result reliability and the identified reasons causing a test case to fail when it was working on a previous release of the software.

Better reliability in the figures measured is achieved by executing the same test suite at least 100 times. RQ3 can’t be answered by studying the system for only a short time period. Data about required modifications are to be collected over a time period not less than 20 weeks.

2.2.1 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 which API calls that are needed, and how much execution time each task consumes, see figure 2.1 for an example of the API request flow to place a bet on one Eurojackpot row.

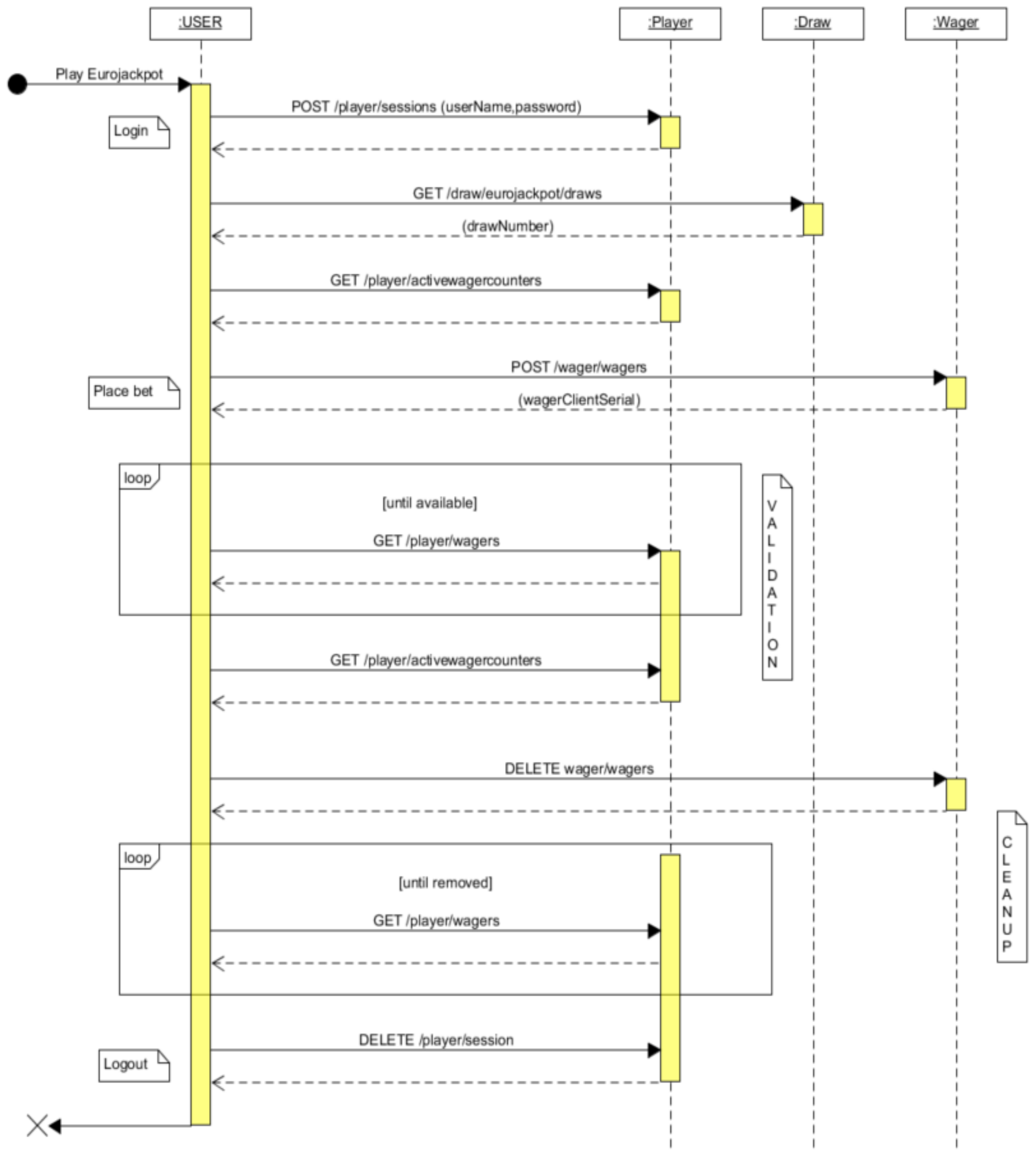


Figure 2.1 API Flow Eurojackpot

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 2.3 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.

For comparable reasons the tasks required to place a bet was grouped into 7 steps, outlined in table 2.2. This was required in order to be able to compare a single API call for example POST wager/wagers, which is used to place a bet, with the navigation of filling in all the marks, selection of payment, ordering payment and finally confirming the payment, which is the processes in the GUI to create the input for the browsers POST wager/wagers request.

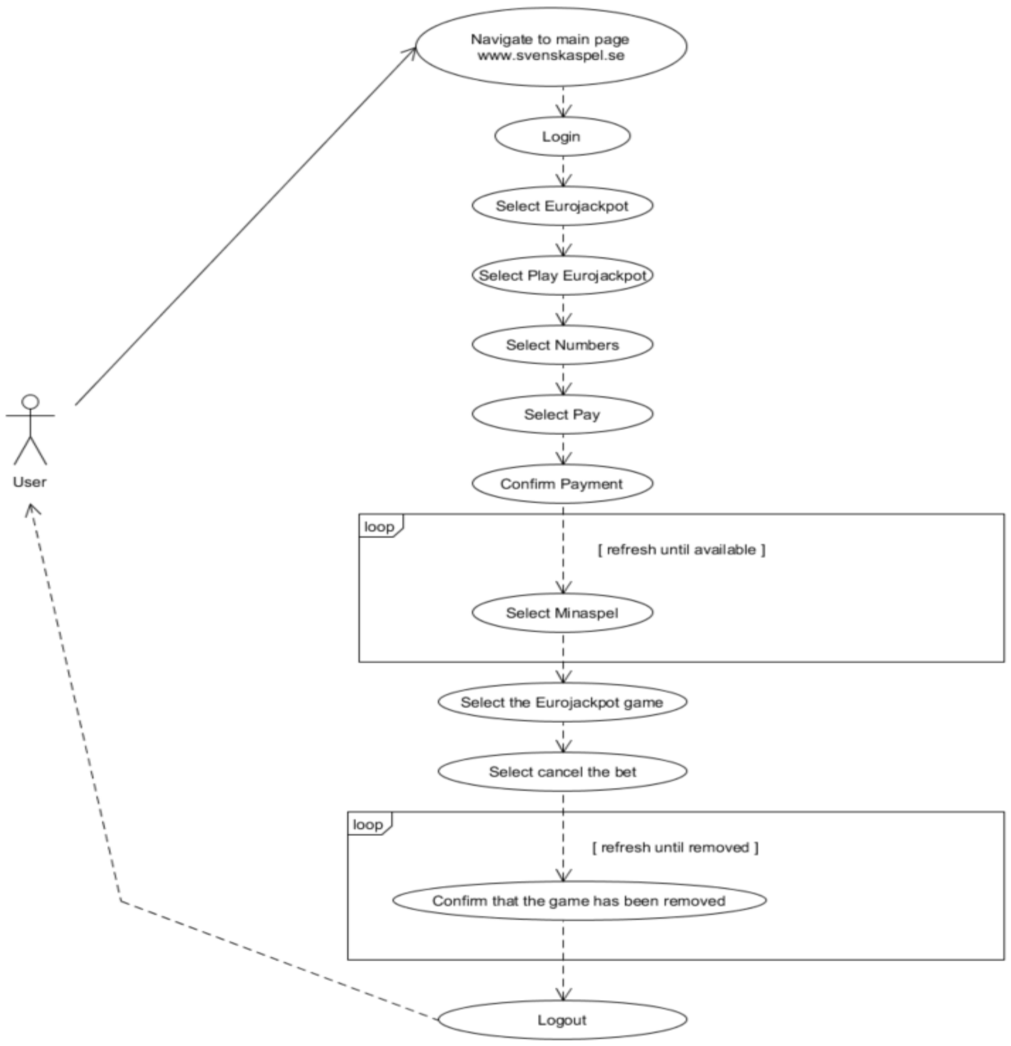
Step	Description	Comment
1	Start a web browser	Not applicable when API calls are used
2	User Login	
3	Initialize the game	
4	Place the bet	
5	Validate placed bet	
6	Remove bet and validate that bet has been removed	
7	User Logout	

Table 2.2 Scenario steps

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 execution times, addressed areas and the modified delta can be considered as answers to research question RQ1.



Navigate to main page www.svenskaspel.se
Login
Select Eurojackpot
Select Play Eurojackpot
Select Numbers
Select Pay
Confirm Payment
User
loop
[refresh until available]
Select Minaspel
Select the Eurojackpot game
Select cancel the bet
loop
[refresh until removed]
Confirm that the game has been removed
Logout

Figure 2.3 GUI Flow Eurojackpot

2.2.2 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.

2.2.3 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.

RQ3 is answered by counting the number of occasions navigation fails due to elements having moved or been replaced causing the sequence to place the bet to fail.

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 once 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.

The upkeep of software versions is not considered as input to RQ3.

2.3 Reliability and Validity

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.x were used during the experimentation.

2.4 Ethical Considerations

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

3 Implementation

In the project a number of different deliverables were implemented.

Primarily a set of test cases for testing of the business requirement related to 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 or to cope with different features of a tool. These were unwanted options that we removed by making an own implementation of a test loader FIA, see figure 3.1.

FIA is a platform independent JAVA implementation.

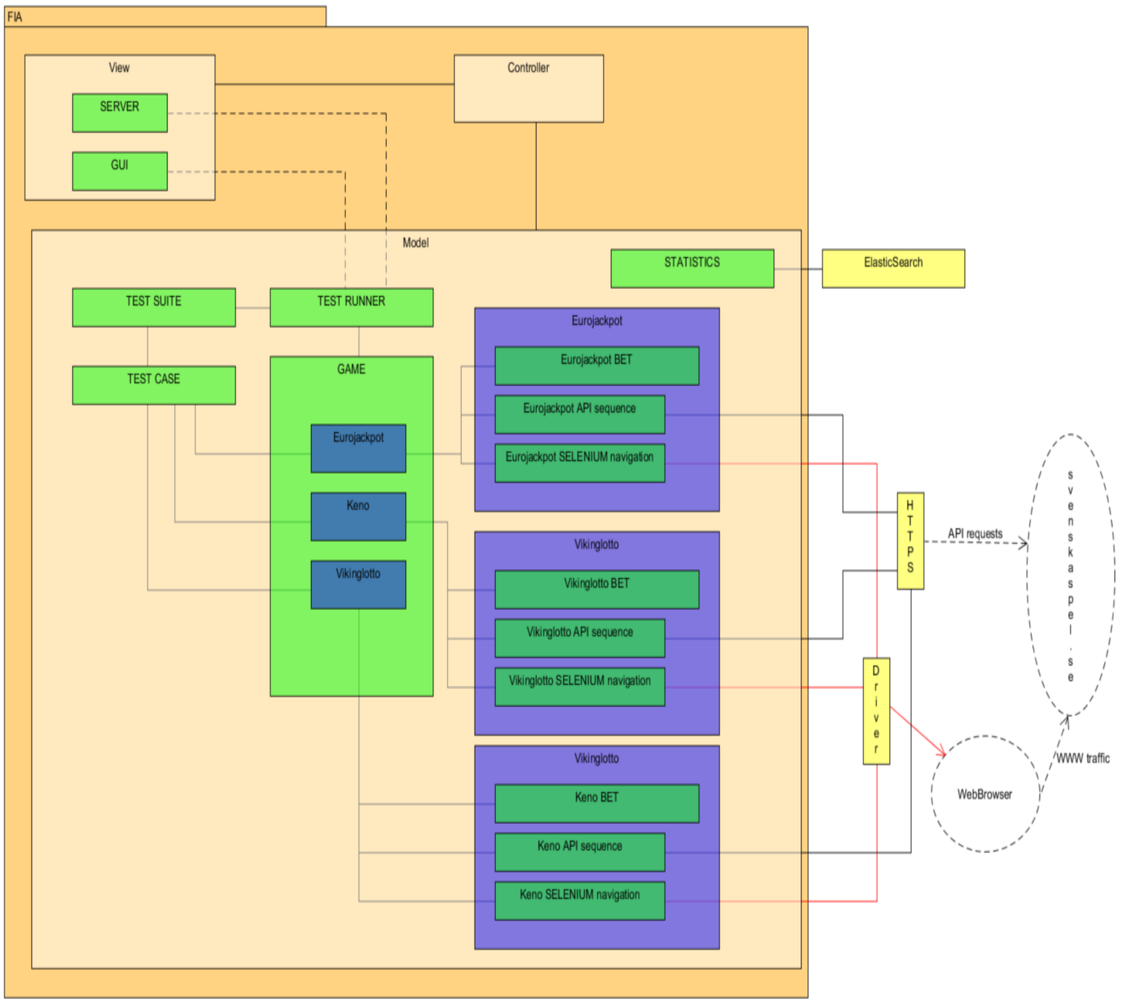


Figure 3.1 FIA overview

3.1 Components of FIA

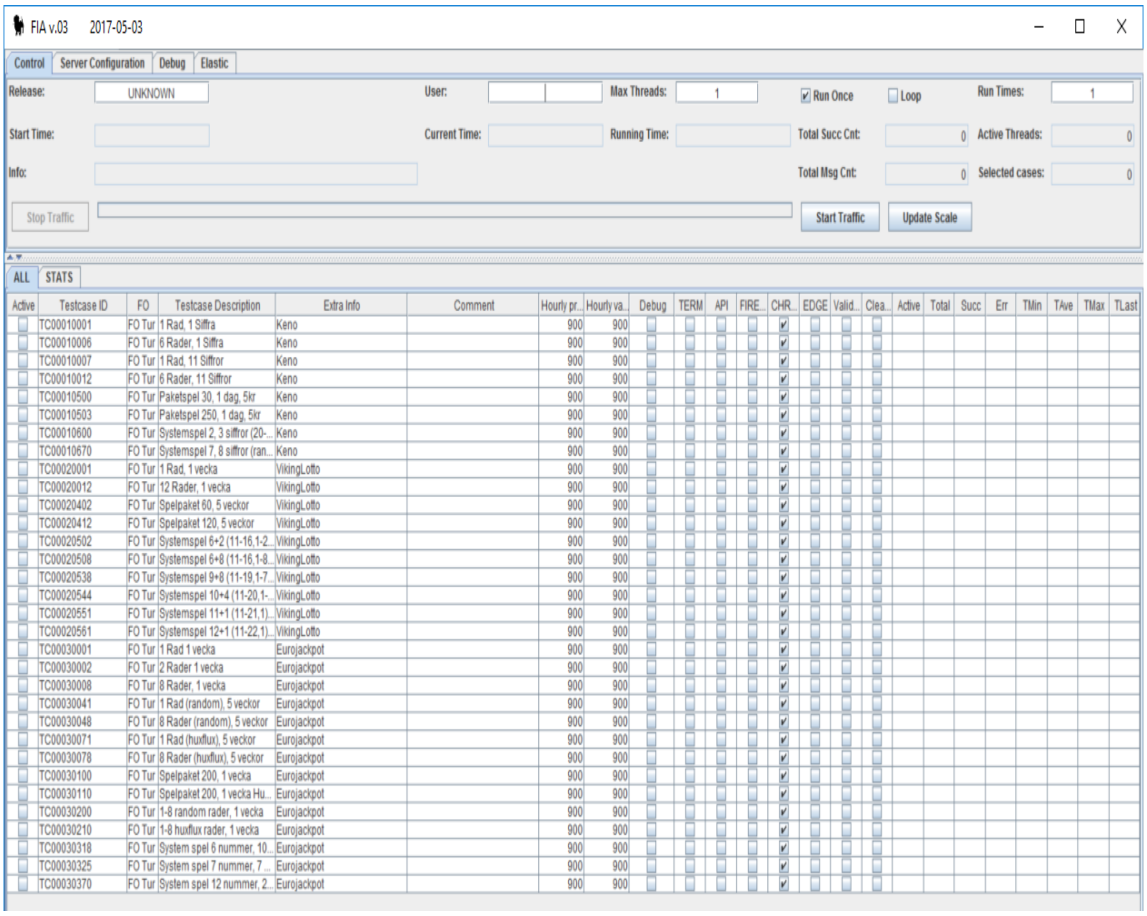
The components of Fia are divided into a Model-View-Controller (MVC) pattern.

3.1.1 View

FIA runs in two different modes, either with a GUI, shown in 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, and it would run on any operating system supporting JAVA.

However Edge or Internet Explorer (IE) are 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.



Release: UNKNOWN
User:
Max Threads: 1
☑ Run Once
☐ Loop
Run Times: 1
Start Time:
Current Time:
Running Time:
Total Succ Cnt: 0
Active Threads: 0
Info:
Total Msg Cnt: 0
Selected cases: 0

Active	Testcase ID	FO	Testcase Description	Extra Info	Comment	Hourly pr...	Hourly va...	Debug	TERM	API	FIRE...	CHR...	EDGE	Valid...	Clea...	Active	Total	Succ	Err	TMin	TAve	TMax	TLast
☐	TC00010001	FO Tur	1 Rad, 1 Siffra	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010006	FO Tur	6 Rader, 1 Siffra	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010007	FO Tur	1 Rad, 11 Siffror	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010012	FO Tur	6 Rader, 11 Siffror	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010500	FO Tur	Paketspel 30, 1 dag, 5kr	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010503	FO Tur	Paketspel 250, 1 dag, 5kr	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010600	FO Tur	Systemspel 2, 3 siffror (20-...	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010670	FO Tur	Systemspel 7, 8 siffror (ran...	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020001	FO Tur	1 Rad, 1 vecka	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020012	FO Tur	12 Rader, 1 vecka	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020402	FO Tur	Spelpaket 60, 5 veckor	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020412	FO Tur	Spelpaket 120, 5 veckor	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020502	FO Tur	Systemspel 6+2 (11-16,1-2...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020508	FO Tur	Systemspel 6+8 (11-16,1-8...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020538	FO Tur	Systemspel 9+8 (11-19,1-7...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020544	FO Tur	Systemspel 10+4 (11-20,1-...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020551	FO Tur	Systemspel 11+1 (11-21,1)...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020561	FO Tur	Systemspel 12+1 (11-22,1)...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030001	FO Tur	1 Rad 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030002	FO Tur	2 Rader 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030008	FO Tur	8 Rader, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030041	FO Tur	1 Rad (random), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030048	FO Tur	8 Rader (random), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030071	FO Tur	1 Rad (huxflux), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030078	FO Tur	8 Rader (huxflux), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030100	FO Tur	Spelpaket 200, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030110	FO Tur	Spelpaket 200, 1 vecka Hu...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030200	FO Tur	1-8 random rader, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030210	FO Tur	1-8 huxflux rader, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030318	FO Tur	System spel 6 nummer, 10...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030325	FO Tur	System spel 7 nummer, 7 ...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030370	FO Tur	System spel 12 nummer, 2...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								

Figure 3.2 GUI of FIA

3.1.2 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 see code 3.3.

```
// CREATE SINGLE INSTANCE

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

// INCREASE ACTIVE THREADS
stats.incThreads();

// START THREAD

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

Code 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 stared.

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:

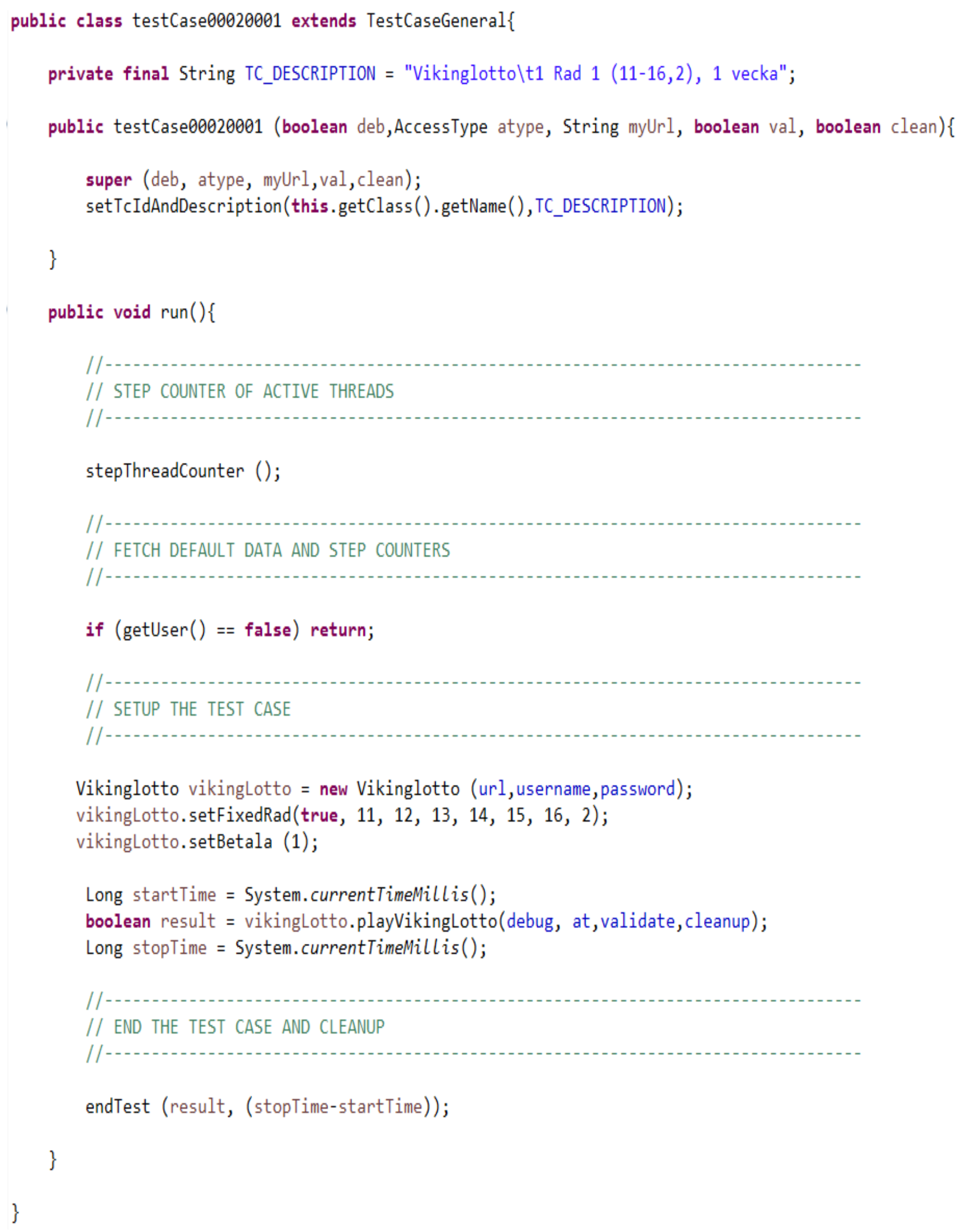
1. Case by Case.
   One test case at a time is executed. The following test case is started upon completion of the previous one.
2. Maximum number of Thread.
   The test cases run in parallel mode up to the maximum number of allowed concurrent test cases.
3. Hourly Traffic profile.
   A required amount 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 any of the 3 currently implemented browsers (Firefox, Chrome and Internet Explorer).

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 of the test case, 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 is shown in code 3.4.



```java
public class testCase00020001 extends TestCaseGeneral{

    private final String TC_DESCRIPTION = "Vikinglotto\t1 Rad 1 (11-16,2), 1 vecka";

    public testCase00020001 (boolean deb,AccessType atype, String myUrl, boolean val, boolean clean){

        super (deb, atype, myUrl,val,clean);
        setTcIdAndDescription(this.getClass().getName(),TC_DESCRIPTION);

    }

    public void run(){

        //-------------------------------------------------------------------------------
        // STEP COUNTER OF ACTIVE THREADS
        //-------------------------------------------------------------------------------

        stepThreadCounter ();

        //-------------------------------------------------------------------------------
        // FETCH DEFAULT DATA AND STEP COUNTERS
        //-------------------------------------------------------------------------------

        if (getUser() == false) return;

        //-------------------------------------------------------------------------------
        // SETUP THE TEST CASE
        //-------------------------------------------------------------------------------

       Vikinglotto vikingLotto = new Vikinglotto (url,username,password);
       vikingLotto.setFixedRad(true, 11, 12, 13, 14, 15, 16, 2);
       vikingLotto.setBetala (1);

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

        //-------------------------------------------------------------------------------
        // END THE TEST CASE AND CLEANUP
        //-------------------------------------------------------------------------------

        endTest (result, (stopTime-startTime));

    }

}
```

Code 3.4 Test Case

Implementation of the Game components
Each game is implemented in its own set of core components see figure 3.5.

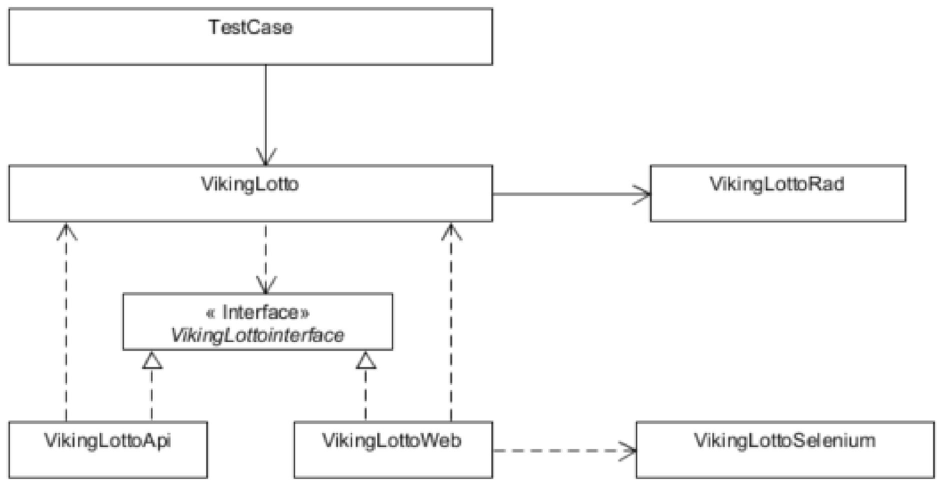


Figure 3.5 Game components

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 see figure 3.6, the only reason behind this decision is to make the implementation of each API as independent as possible see 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.

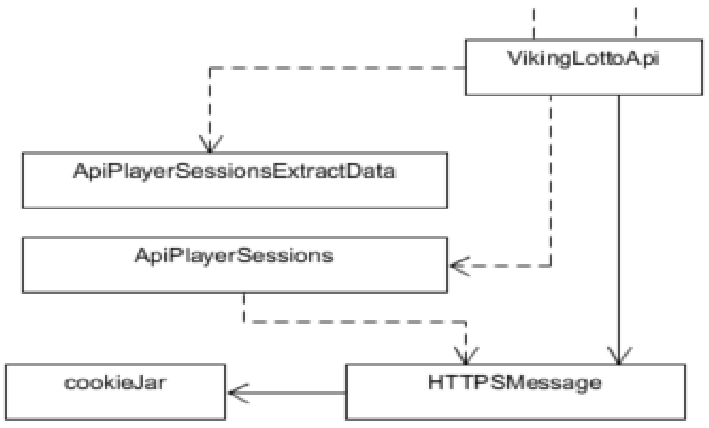


Figure 3.6 API Message sending

```
//------------------------------------------------------------
// 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 see 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.

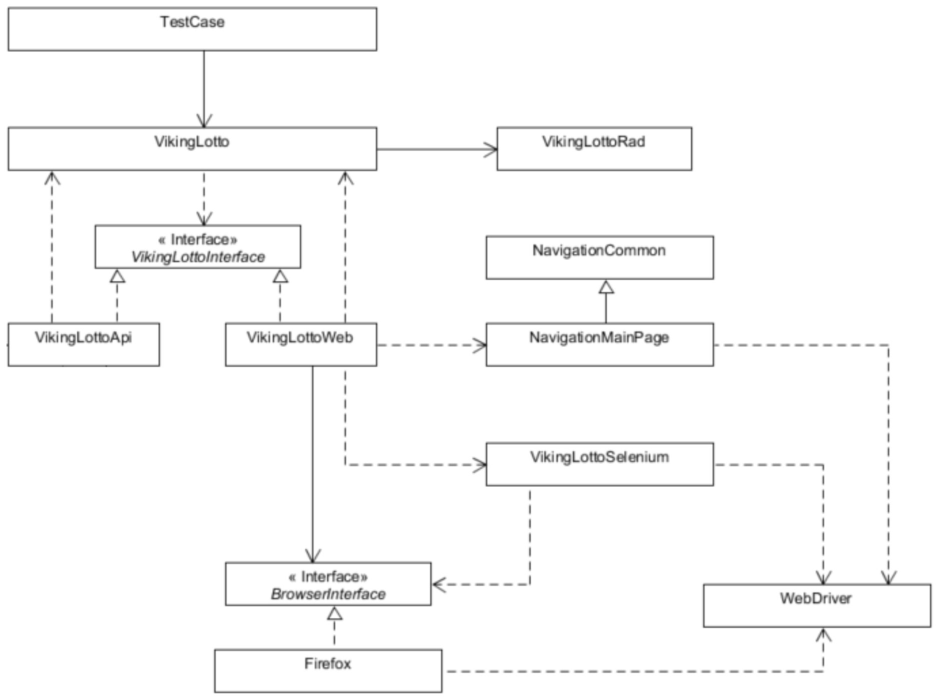


Figure 3.8 Web navigation

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 see 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 (handing 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 see figure 3.10. The core components which are specific for the game are the only ones replaced.

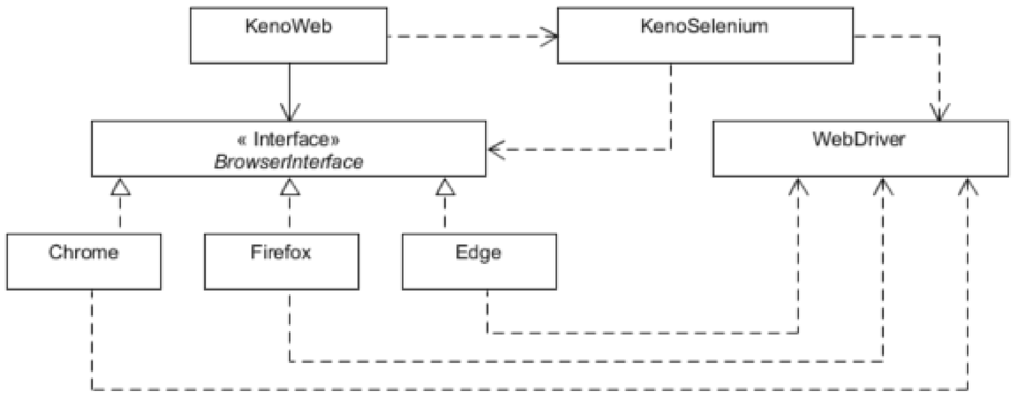


Figure 3.9 Web browsers

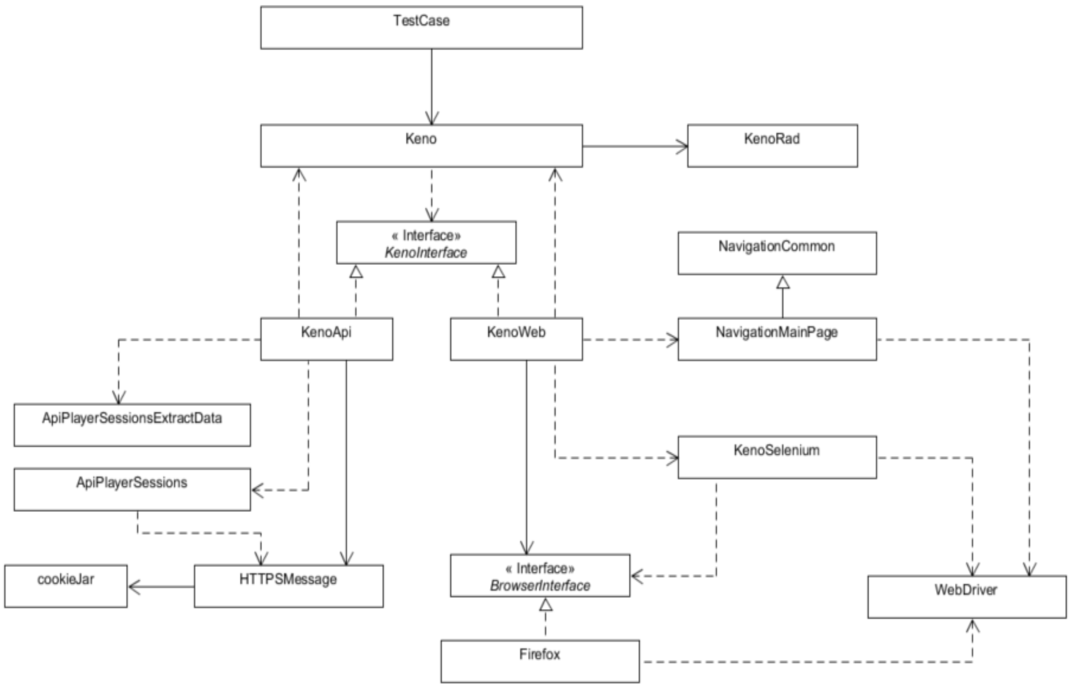


Figure 3.10 Keno Overview

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

The selected test cases are a subset of the total test suite. The selection 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, from a GUI point of view.

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: Selected Test cases Keno

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

Table 4.2: Selected Test cases Vikinglotto

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 4.3: Selected Test cases Eurojackpot

For a comparable analysis between the usage of direct API calls and GUI navigation using a web browser, the scenarios are divided into 7 steps, listed in table 2.2.

4.1 Test case execution times

Test case	API Execution Time (ms)	Selenium Execution Time (ms)
TC00010001	2082,16	21176,29
TC00010006	2045,30	24230,83
TC00010007	1984,57	22142,45
TC00010012	2102,24	32648,59
TC00010500	2052,89	19344,95
TC00010503	2062,64	19771,95
TC00010600	2137,88	19635,88
TC00010670	2041,09	20317,67
TC00020001	2016,32	21229,98
TC00020012	1998,18	43713,64
TC00020402	2036,84	19212,04
TC00020412	2031,02	19476,82
TC00020502	1977,10	19863,34
TC00020508	2019,29	20362,11
TC00020538	2029,55	20670,02
TC00020544	1994,33	20236,41
TC00020551	2015,50	20320,83
TC00020561	2042,08	20676,85
TC00030001	2036,11	25045,28
TC00030002	1986,22	32009,53
TC00030008	2049,16	73860,56
TC00030041	2033,23	25427,15
TC00030048	2000,31	75222,10
TC00030071	2057,24	21255,97
TC00030078	2100,98	54211,40
TC00030100	2043,81	19355,83
TC00030110	2033,88	19361,66
TC00030200	2018,48	45848,29
TC00030210	2042,44	34839,32
TC00030318	2032,28	20696,83
TC00030325	2016,10	20250,69
TC00030370	2042,21	20657,99

Table 4.4: Average execution times of the test case for 100 test runs

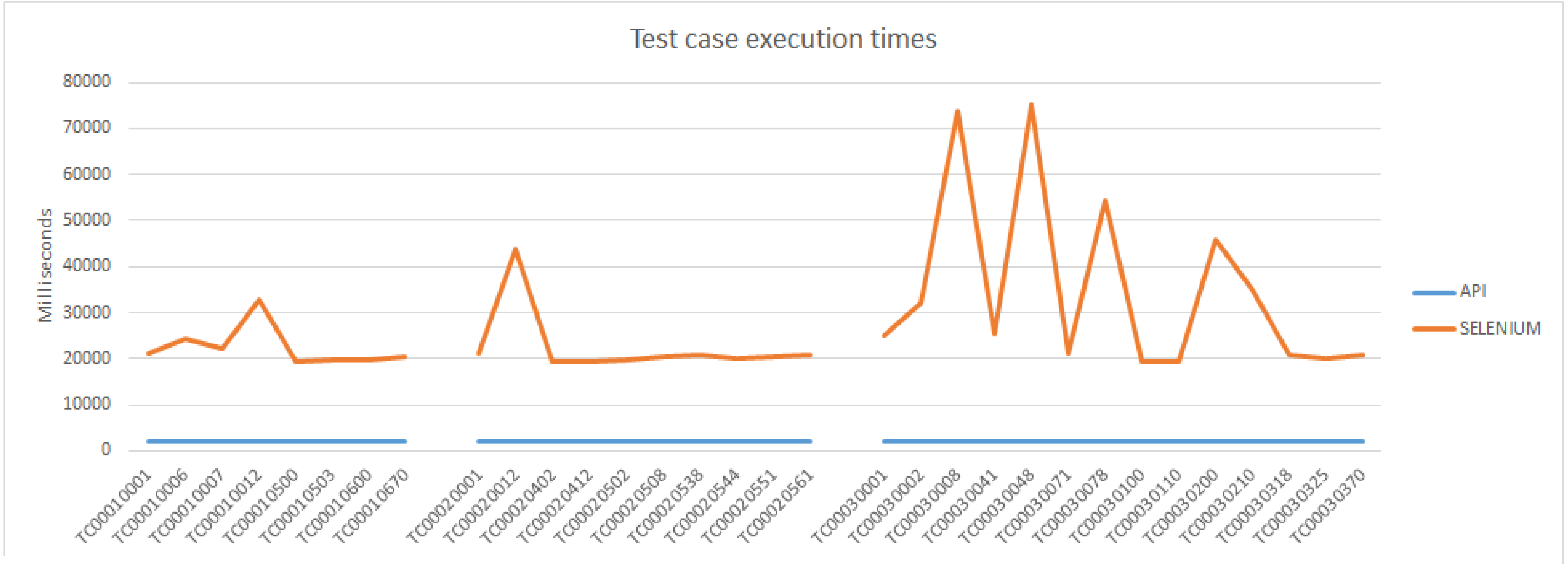


Figure 4.5: Test case execution times for 100 test runs

Game	**Interface**	**Step 1 (ms)**	**Step 2 (ms)**	**Step 3 (ms)**	**Step 4 (ms)**
Keno	Api	*	53,08	97,36	30,67
Keno	Selenium	3378,64	2747,26	1481,44	4820,89
Vikinglotto	Api	*	35,43	116,79	29,15
Vikinglotto	Selenium	3402,94	2768,62	1445,36	4021,34
Eurojackpot	Api	*	38,10	131,14	30,39
Eurojackpot	Selenium	3458,63	2771,23	1380,59	16663,62

Table 4.6: Average execution times for scenario steps 1-4 for 100 test runs
* Not applicable

Game	**Interface**	**Step 5 (ms)**	**Step 6 (ms)**	**Step 7 (ms)**
Keno	Api	973,02	839,05	59,36
Keno	Selenium	1810,65	3479,70	226,34
Vikinglotto	Api	976,39	813,8	34,74
Vikinglotto	Selenium	2517,41	3411,33	231,86
Eurojackpot	Api	1055,87	761,06	40,81
Eurojackpot	Selenium	2327,34	3705,35	230,93

Table 4.7: Average execution times for scenario steps 5-7 for 100 test runs
* Not applicable

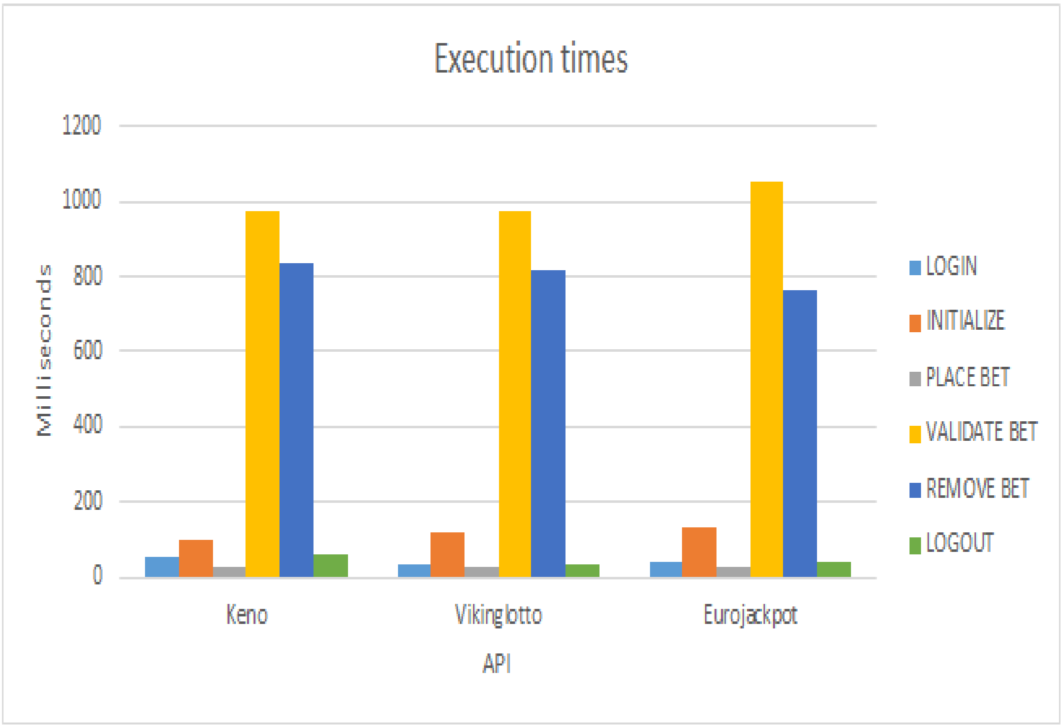


Figure 4.8 Execution times API

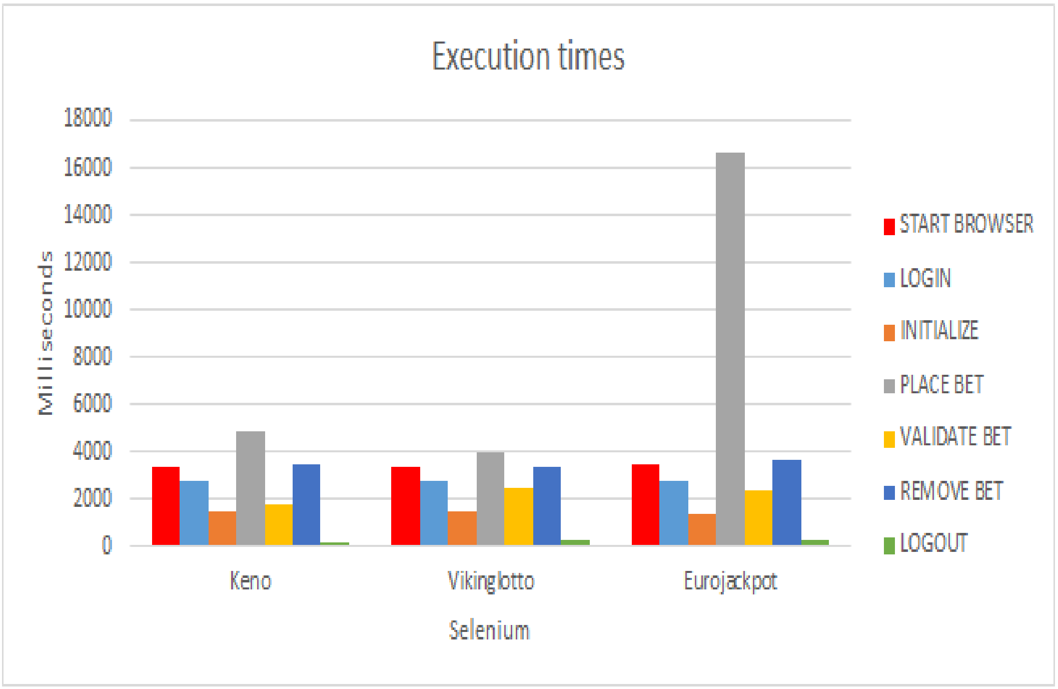


Figure 4.9 Execution times Selenium

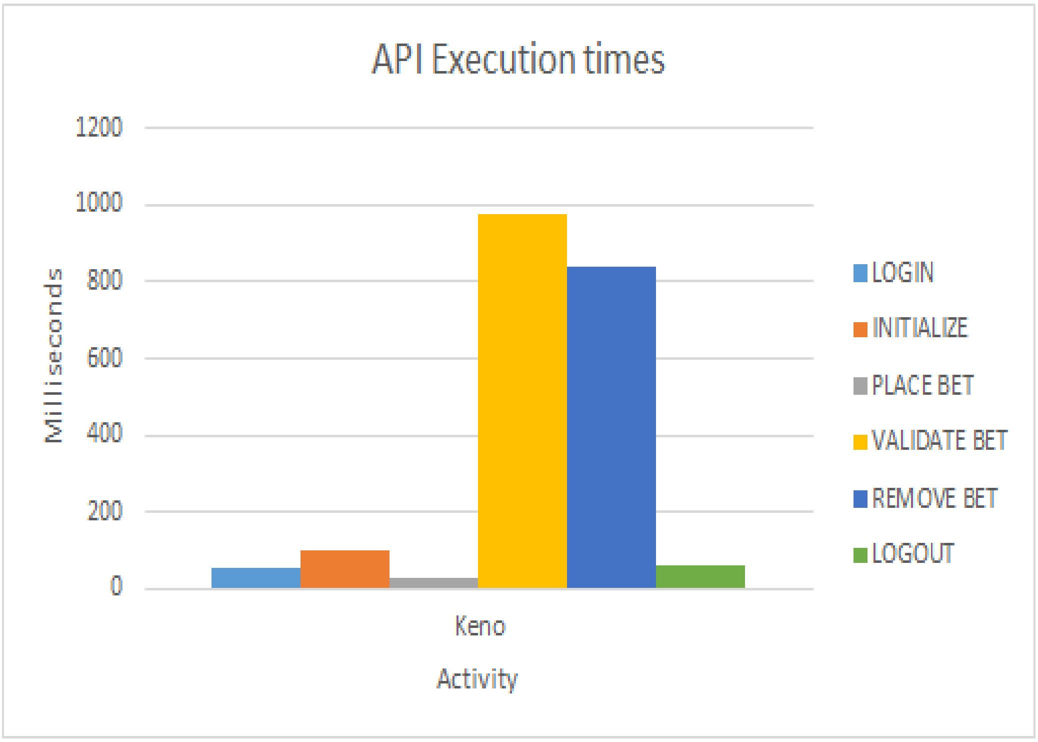


Figure 4.10 Keno API execution times

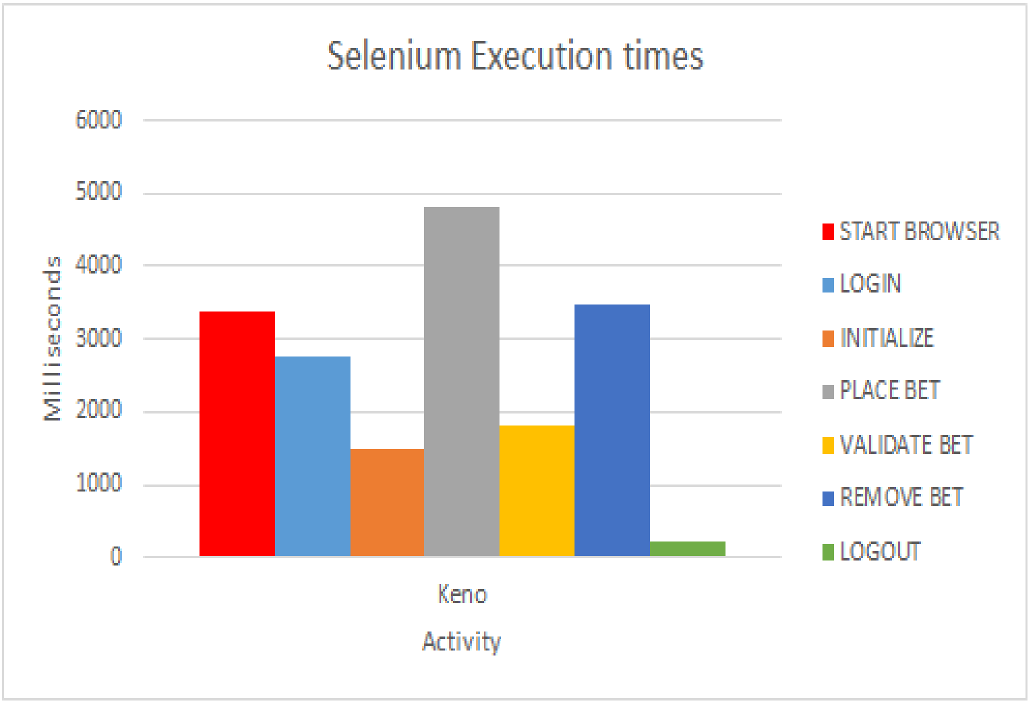


Figure 4.11 Keno Selenium execution times

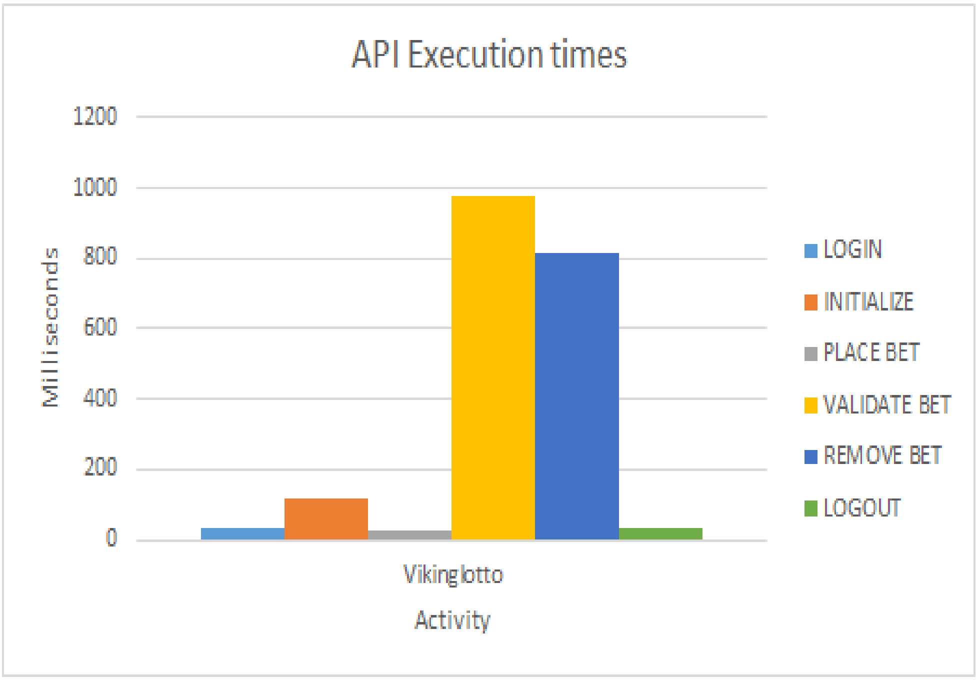


Figure 4.12 Vikinglotto API execution times

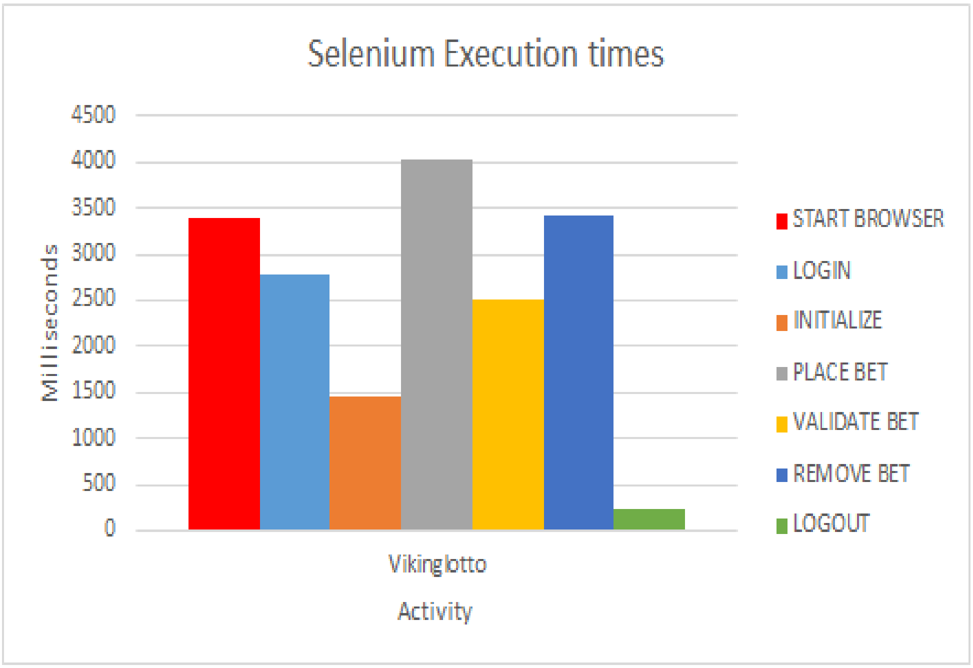


Figure 4.13 Vikinglotto Selenium execution times

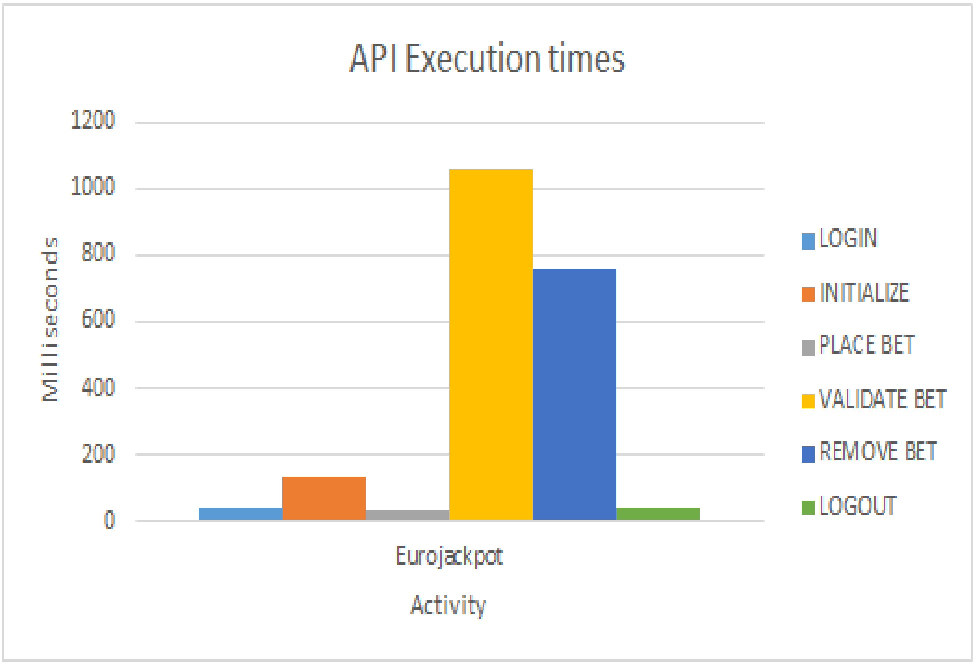


Figure 4.14 Eurojackpot API execution times

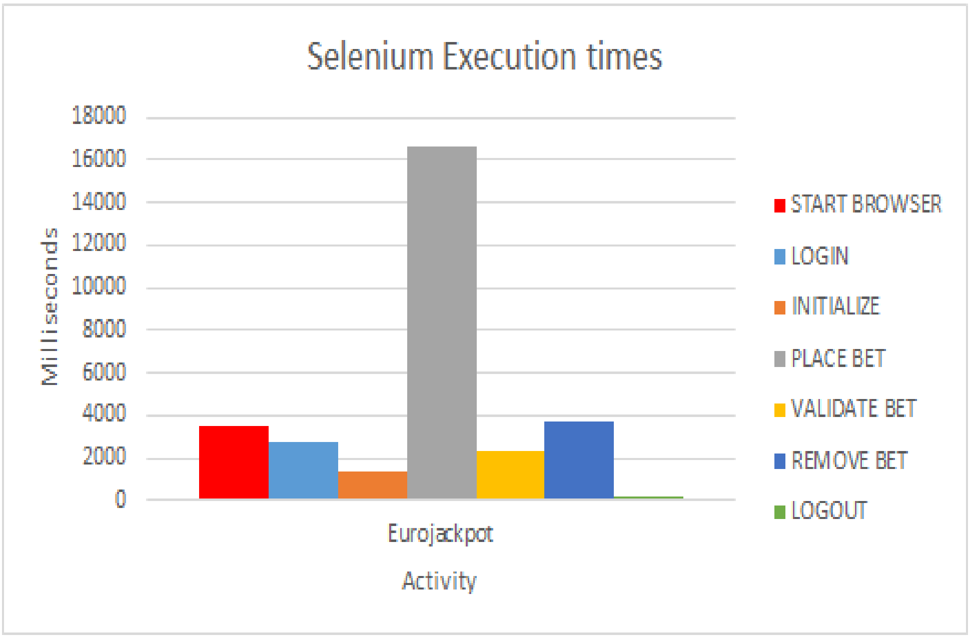


Figure 4.15 Eurojackpot Selenium execution times

Scroll into view (ms)	Set value (ms)	Set kung keno (ms)
122,82	154,22	163,38

Table 4.16: Setting Keno values Average execution times from 100 test executions

Scroll into view (ms)	Set value (ms)
147,78	161,41

Table 4.17: Setting Vikinglotto values Average execution times from 100 test executions

Wait for random values (ms)	Scroll into view (ms)	Select wheel (ms)	Set wheel (ms)
1782,80	29,57	65,19	95,97

Table 4.18: Setting Eurojackpot values Average execution times from 100 test executions

Game	Step 2 (ms)	Step 3 (ms)	Step 4 (ms)
Keno	50,63	63,11	27,23
Vikinglotto	33,08	56,76	26,36
Eurojackpot	35,45	90,00	27,36

Table 4.19: Detailed API execution times Step 2-4 (per API call) from 100 test executions

Game	Step 5 (ms)	Step 6 (ms)	Step 7 (ms)
Keno	28,37	28,23 + 28,37	56,98
Vikinglotto	25,93	26,81 + 25,93	32,43
Eurojackpot	27,13	28,30 + 27,13	38,25

Table 4.20: Detailed API execution times Step 5-7 from 100 test executions

4.2 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 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 4.21: Average execution result over time for Keno

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

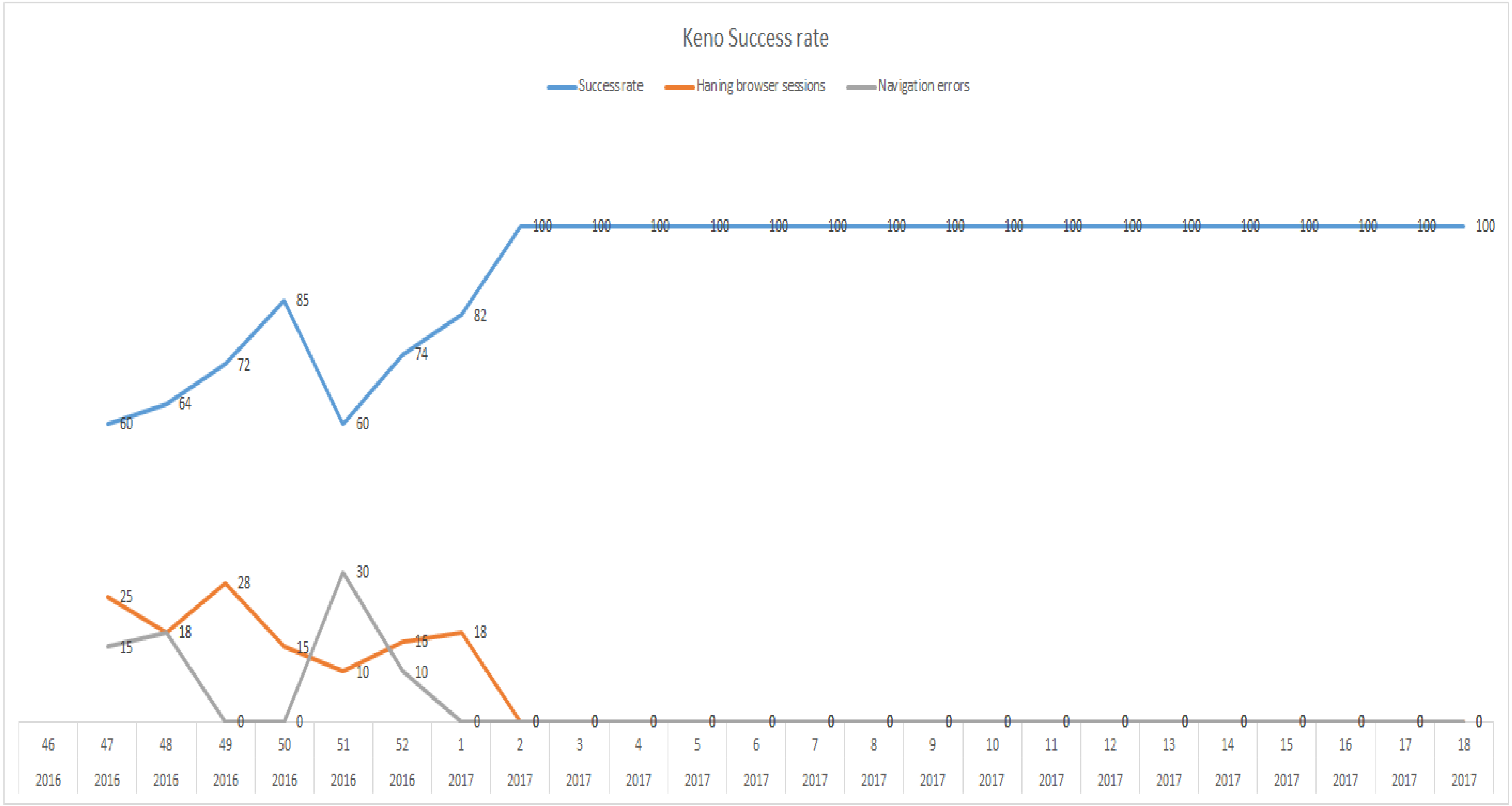


Figure 4.22: Keno success rate over time

4.3.2 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 4.23: Average execution result over time for Vikinglotto

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

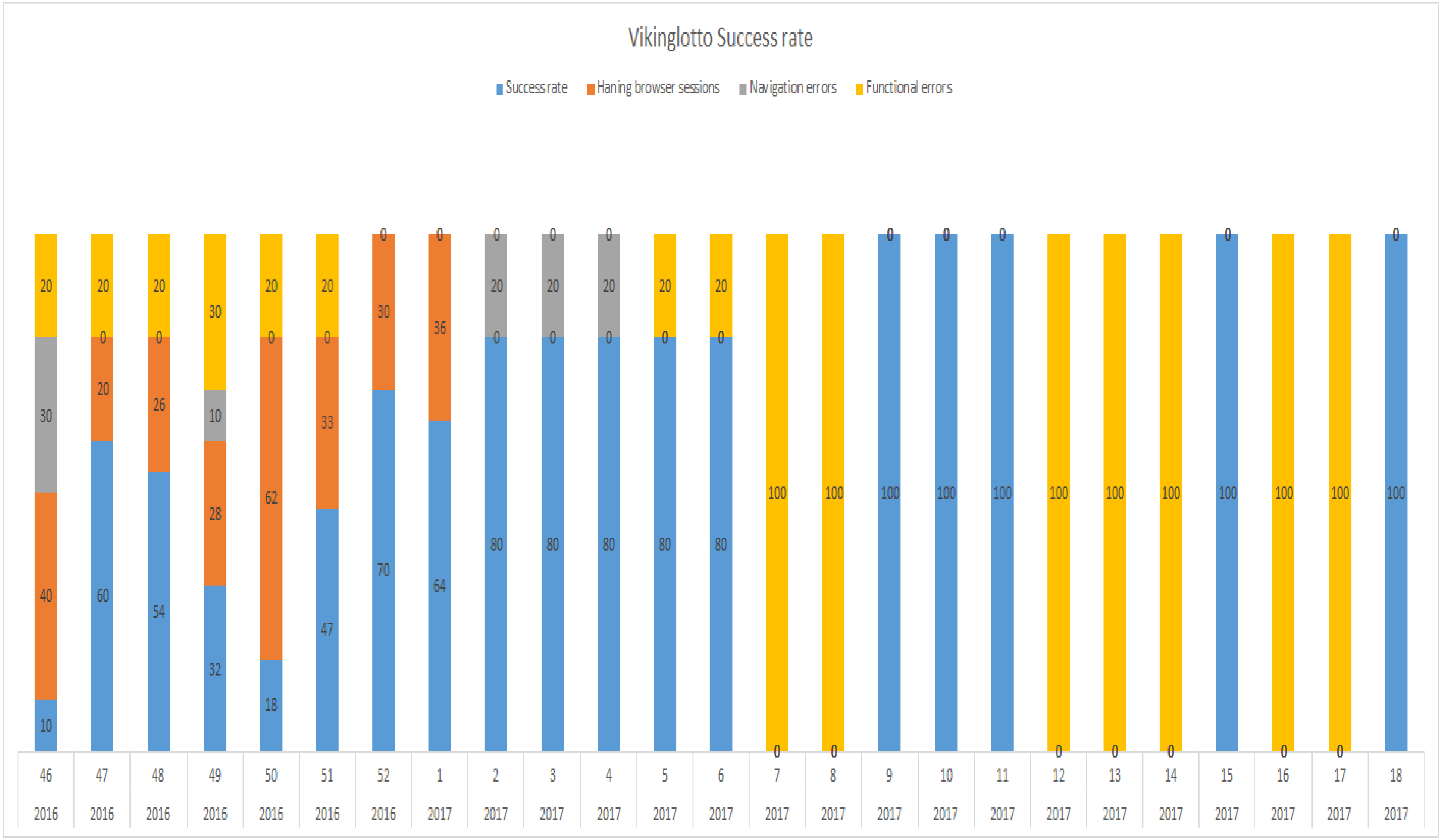


Figure 4.24: Vikinglotto sucess rate over time

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 4.25: Average execution result over time for Eurojackpot

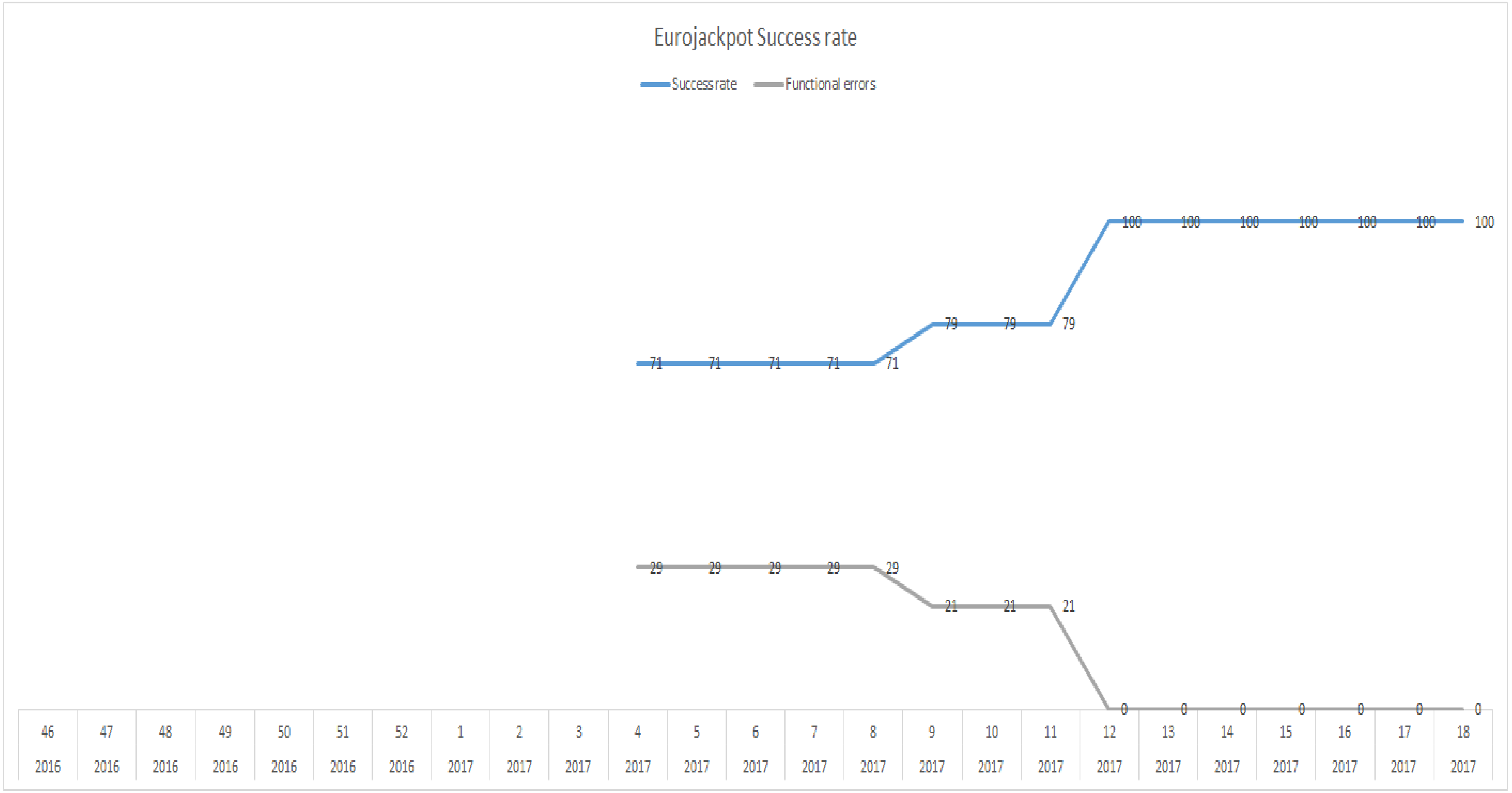


Figure 4.26: Eurojackpot success rate over time

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

4.3 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 the only new games that was designed during the time period when the research was conducted.

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 4.27: Number of times GUI navigation was updated

5 Analysis

Analysis of the test result was broken down in three different categories, execution times, flakiness and finally issues causing test cases to break.

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 times being spent on placing the bet is very small. This can be seen in table 4.4 and figure 4.5 where the test execution times is observed to be close to 2 seconds per test case regardless of which type bet that is placed and game that is played.

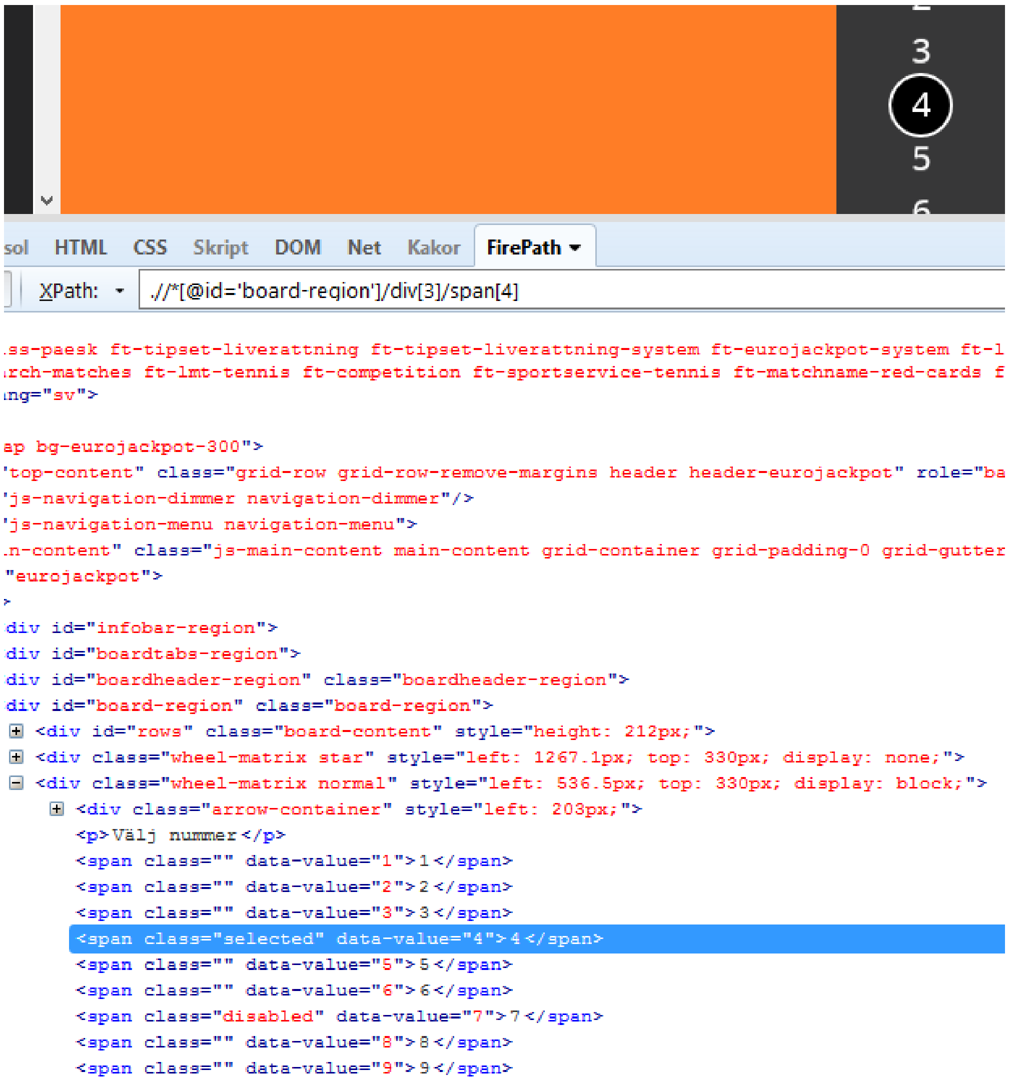
The difference in execution times is between 1977.10 and 2137.88 milliseconds, a difference with 160.78 milliseconds. The small difference is derived mainly from two sources. The time taken by the Ethernet network to send and retrieve the request as well as the internal delay between placed bet and until it is possible to query the bet. 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 consumed in the API sequence flow is spent in the validation part. I.e. validating that the bet has been placed and validating that the bet has been removed (figures 4.8 – 4.15).

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 (figures 4.8-4.15).

The differences in the navigation are in the part where the bet is being placed. Which mainly is affected by the number of elements that needs to be selected. Also the visual effects of the game needs to be taken into account. In the Keno and Vikinglotto game there are no significant visual effects when requesting the game to be played. However in the Eurojackpot game there is an effect which randomizes the initial values. This is partly the reason behind the significantly longer time spent placing a Eurojackpot game compared to Keno and Vikinglotto.

The time spent on placing a game is also affected by the pattern implementation of locating the web element and scrolling the element into view prior to selecting the element. While this adds to removal of flaky tests there is a time penalty on placing the game as between 29.57 – 147.78 milliseconds (figure 4.16-4.18) are used in average on moving the object into a view where it is selectable. Selecting a single object takes between 154.22 – 161.41 milliseconds (figure 4.16-4.18). Hence each game requires a few seconds just to place the bet.

The bigger time difference placing a Eurojackpot game is not only related to waiting for the effect displaying the random wheels. It is also due to how data can be read in a reliable way. The trustable way to read the value of each wheel is to access the wheel and check which digit is selected for the particular wheel is to read the attribute value of the class for each wheel, see figure 5.1.



```
XPath:  .//*[@id='board-region']/div[3]/span[4]
```

```
.ss-paesk ft-tipset-liverattning ft-tipset-liverattning-system ft-eurojackpot-system ft-l
.rch-matches ft-lmt-tennis ft-competition ft-sportservice-tennis ft-matchname-red-cards f
.ng="sv">

ap bg-eurojackpot-300">
'top-content" class="grid-row grid-row-remove-margins header header-eurojackpot" role="ba
'js-navigation-dimmer navigation-dimmer"/>
'js-navigation-menu navigation-menu">
.n-content" class="js-main-content main-content grid-container grid-padding-0 grid-gutter
"eurojackpot">
>
div id="infobar-region">
div id="boardtabs-region">
div id="boardheader-region" class="boardheader-region">
div id="board-region" class="board-region">
 ⊞ <div id="rows" class="board-content" style="height: 212px;">
 ⊞ <div class="wheel-matrix star" style="left: 1267.1px; top: 330px; display: none;">
 ⊟ <div class="wheel-matrix normal" style="left: 536.5px; top: 330px; display: block;">
    ⊞ <div class="arrow-container" style="left: 203px;">
      <p>Välj nummer</p>
      <span class="" data-value="1">1</span>
      <span class="" data-value="2">2</span>
      <span class="" data-value="3">3</span>
      <span class="selected" data-value="4">4</span>
      <span class="" data-value="5">5</span>
      <span class="" data-value="6">6</span>
      <span class="disabled" data-value="7">7</span>
      <span class="" data-value="8">8</span>
      <span class="" data-value="9">9</span>
```

Figure 5.1 Wheel value locator

Other differences adding up in the longer execution times is the validation of a placed bet are navigation to get to the pages for accessing the data. The approximately one second delay in being able to access the placed bet information as seen in the API cases (table 4.7) resulted in a need to at some occasions reload the full mina-spel page. This page was used to validate that the game had been placed in the Selenium case. If the data was not accessible the first time the page is accessed, the whole page needs to be reloaded.

These issues collectively adds up in a significantly longer execution times.

A comparison between the execution times needs to consider that when the web browser is used to place the game significantly more information is requested from the system. Effects like greeting messages, fetching the players name, settings etc. are not required in the API case, however when the GUI is used it is a part of the implementation and thus required.

This makes it difficult to compare the values in table 4.4 and tables 4.16-4.20 by just mapping them one to one.

Also the tradeoff between speed and reliability that is required adds additional execution time to the GUI navigation. This adds up to the difference for placing a single row game with 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 20-75 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 tables 4.21, 4.23 and 4.25 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 4.21, 4.23 and 4.25. 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.

Vikinglotto 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. In table 4.21 it is shown 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 Vikinglotto at a few occasions. In total 15 occasions broke the test case due to some GUI element being replaced (table 4.27).

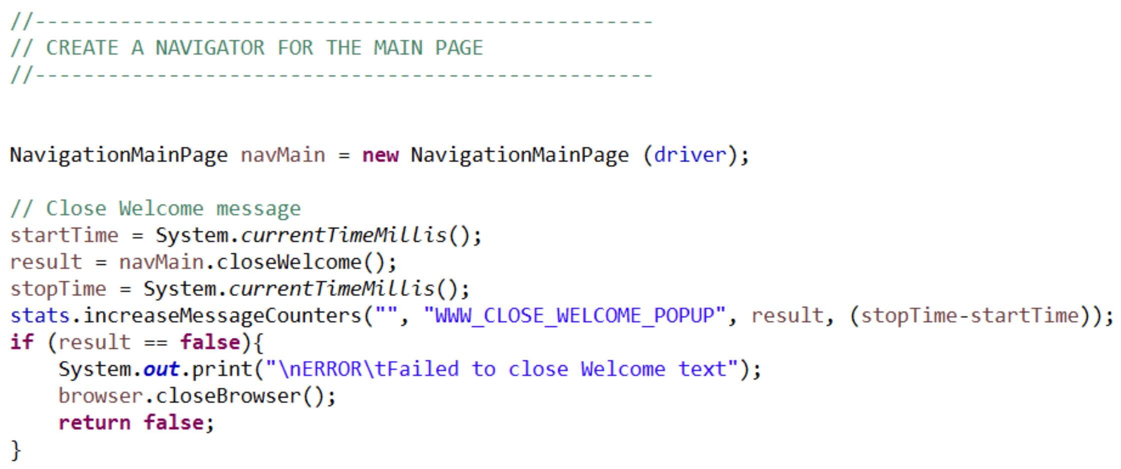
Eurojackpot and Keno were both existing games and not redesigned during the time interval when the data was collected, only Vikinglotto was a new product, hence even though a number of GUI elements are common for all the existing games the navigation only broke on a few occasions.

6 Discussion

The differences in execution times are not easy to compare between different web applications as each comes with its own set of dependencies. Hence a comparison with other research on web applications would mainly be speculative.

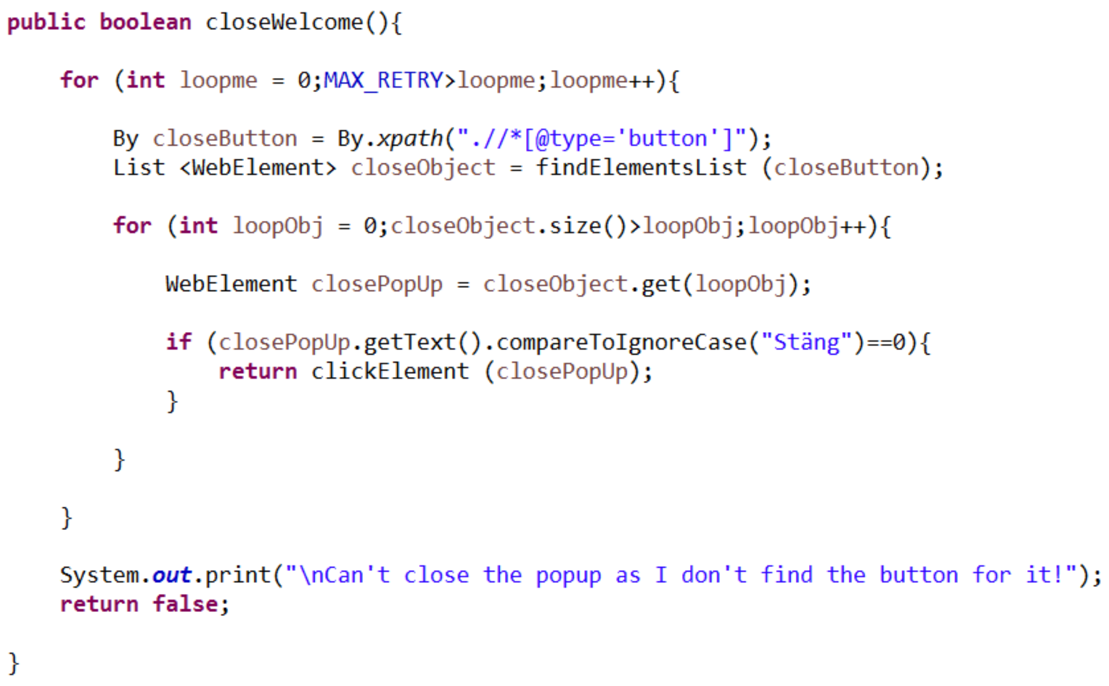
But this research shows that improving test execution times needs to be considered in a wider scope. By tweaking the navigation and with optimization of some parts of the test code used some improvements would be possible, however it would not reduce the effect from the time spent on visualization of objects on the web browser. As an example there is an approximately 3 seconds spent on just retrieving the digits randomly selected for a Eurojackpot row. When it comes to testing of the business requirement of placing a Eurojackpot be this extra delay doesn't contribute to anything. If the values would be readable in advance to the spinning wheels effect the navigation could disregard to the spinning and continue with the execution considerably faster. This shows that already the design needs to take into account how the code will be tested if significant time savings should be achieved.

The navigation implemented doesn't use delays rather it tries to find the following object to navigate on as fast as possible, i.e. when it fulfills the two tasks or, object being located and that the object is scrolled into view so it can be acted upon, see code 6.1 and 6.2.



```
//---------------------------------------------------
// CREATE A NAVIGATOR FOR THE MAIN PAGE
//---------------------------------------------------


NavigationMainPage navMain = new NavigationMainPage (driver);

// Close Welcome message
startTime = System.currentTimeMillis();
result = navMain.closeWelcome();
stopTime = System.currentTimeMillis();
stats.increaseMessageCounters("", "WWW_CLOSE_WELCOME_POPUP", result, (stopTime-startTime));
if (result == false){
    System.out.print("\nERROR\tFailed to close Welcome text");
    browser.closeBrowser();
    return false;
}
```

Code 6.1 Close Welcome popup



```java
public boolean closeWelcome(){

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

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

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

            WebElement closePopUp = closeObject.get(loopObj);

            if (closePopUp.getText().compareToIgnoreCase("Stäng")==0){
                return clickElement (closePopUp);
            }

        }

    }

    System.out.print("\nCan't close the popup as I don't find the button for it!");
    return false;

}
```

Figure 6.2 Close Welcome popup locator

As mentioned the issue needed to be evaluated from a wider perspective however the main issues identified as contributing most to the execution times were:

1. Waiting time for objects to complete visual effects prior to being in a reliable state where interaction with the element was possible.
2. The need to ensure that objects to be interacted on always are in view, where the browser implementation allowed for interaction with the element.

These were the top two issues found during the research and also answered RQ1.

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 (tables 4.21, 4.23 and 4.25).

The remaining navigational errors that sometimes cause a test case to fail was analyzed by looking at the dumped screenshots 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 shown in figures 6.3, 6.4, and 6.5.

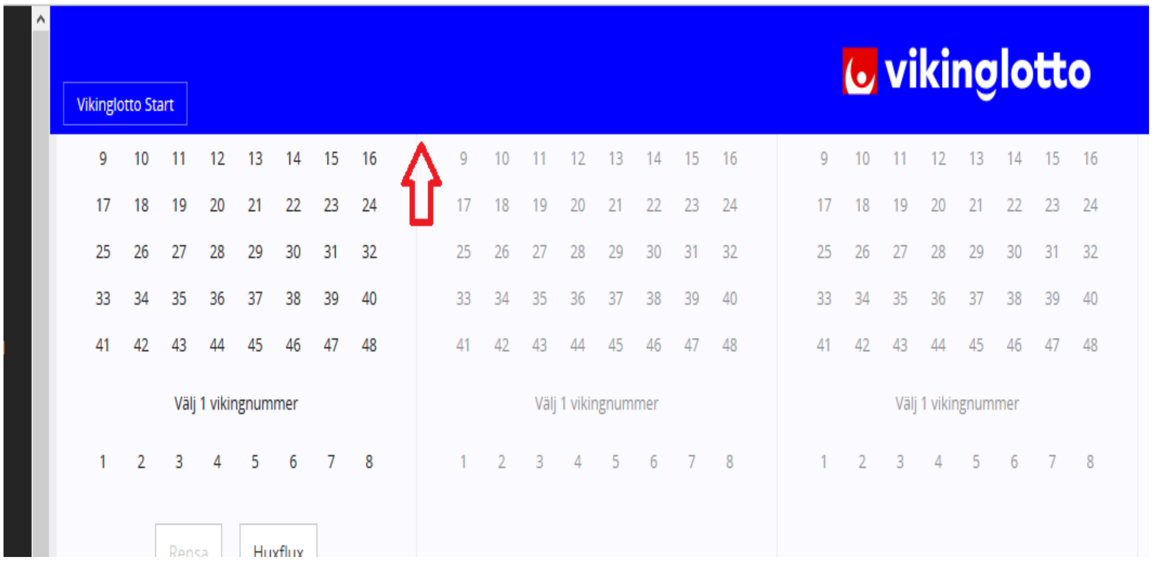


Figure 6.3 Digits 1-8 hidden under the banner

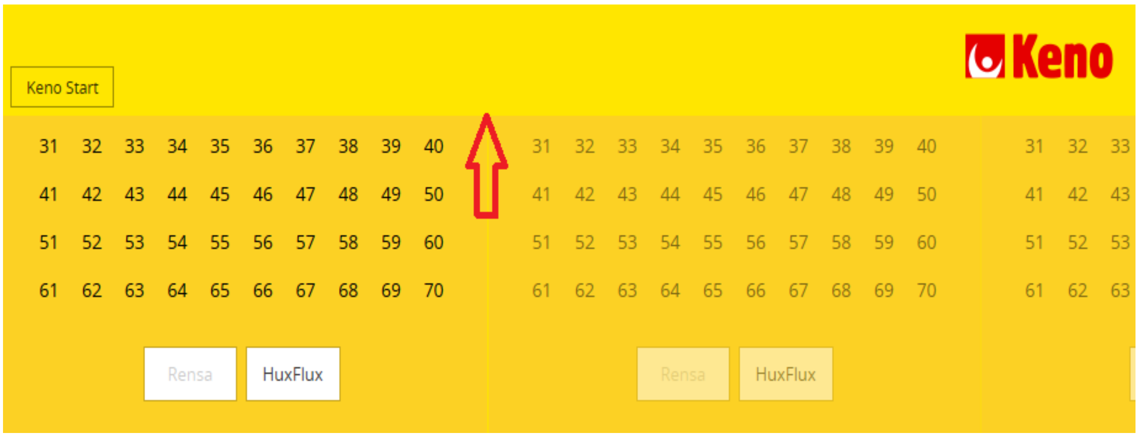
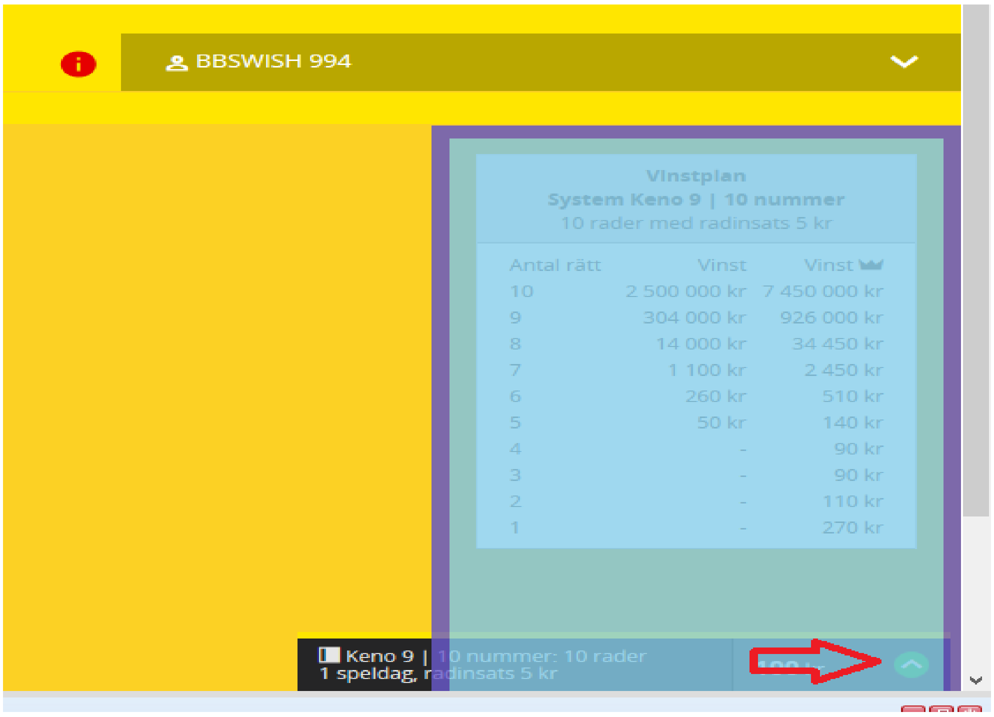


Figure 6.4 Digits 1-30 hidden under the banner



Vinstplan
System Keno 9 | 10 nummer
10 rader med radinsats 5 kr

Antal rätt	Vinst	Vinst
10	2 500 000 kr	7 450 000 kr
9	304 000 kr	926 000 kr
8	14 000 kr	34 450 kr
7	1 100 kr	2 450 kr
6	260 kr	510 kr
5	50 kr	140 kr
4	-	90 kr
3	-	90 kr
2	-	110 kr
1	-	270 kr

Keno 9 | 10 nummer: 10 rader
1 speldag, radinsats 5 kr

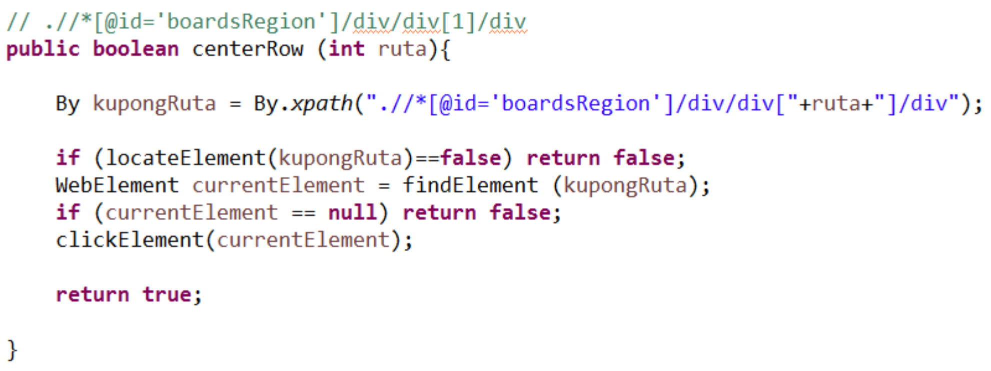
Figure 6.5 Button hidden under the info window

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. The solution is shown in code 6.6, 6.7 and 6.8.

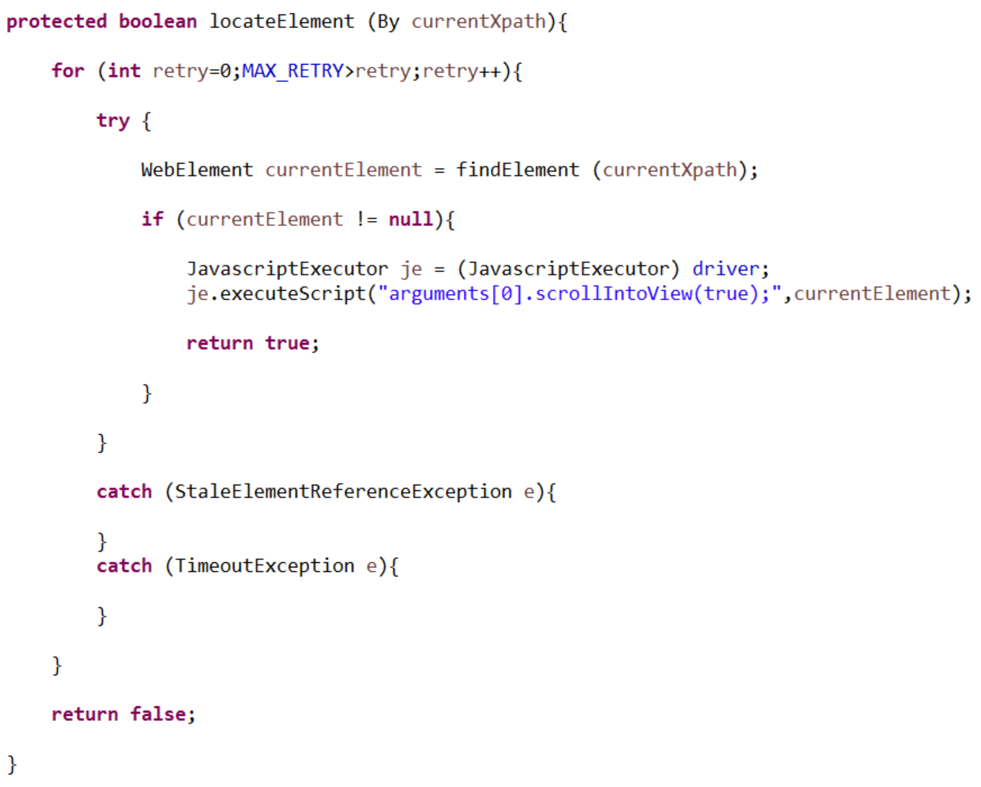
```
public boolean playRow (int ruta, ArrayList <Integer> numbers){

    // 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;

}
```

Code 6.6 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;

}
```

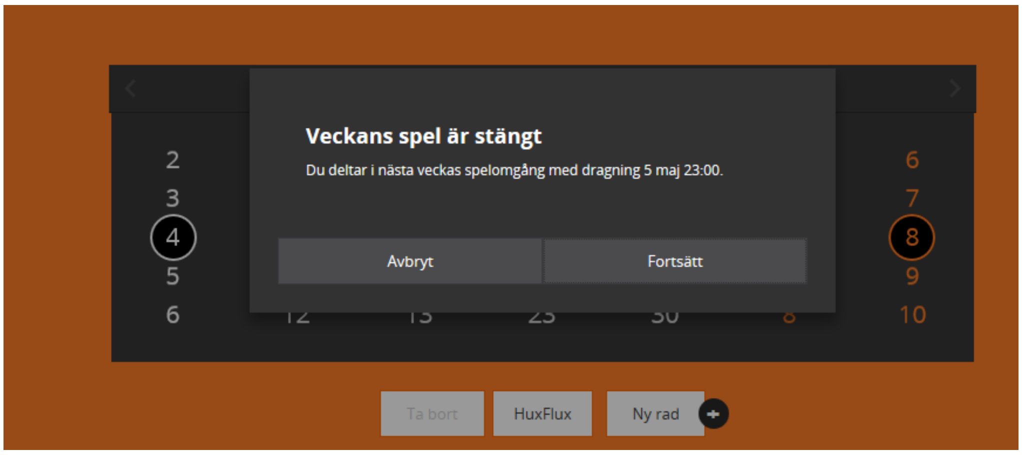
Figure 6.7 centerRow method



```
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;

}
```

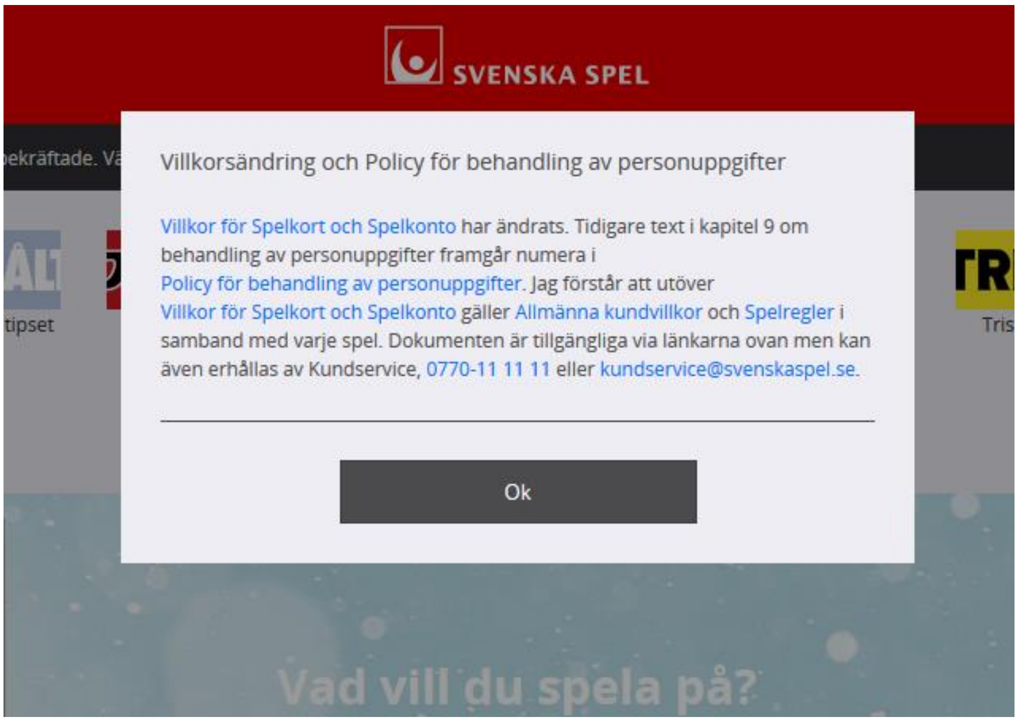
Figure 6.8 locateElement method

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. A few of these unexpected popups are shown in figures 6.9-6.13.


Veckans spel är stängt

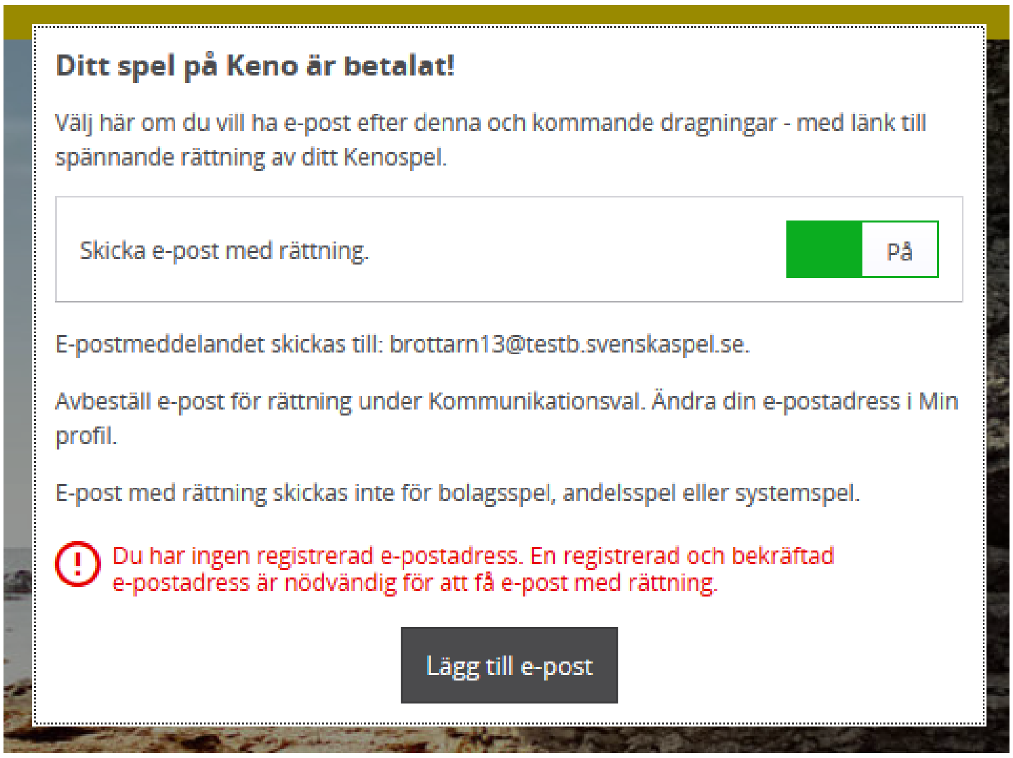
Du deltar i nästa veckas spelomgång med dragning 5 maj 23:00.

Figure 6.9 Popup closed for new bets


Villkorsändring och Policy för behandling av personuppgifter

Villkor för Spelkort och Spelkonto har ändrats. Tidigare text i kapitel 9 om behandling av personuppgifter framgår numera i Policy för behandling av personuppgifter. Jag förstår att utöver Villkor för Spelkort och Spelkonto gäller Allmänna kundvillkor och Spelregler i samband med varje spel. Dokumenten är tillgängliga via länkarna ovan men kan även erhållas av Kundservice, 0770-11 11 11 eller kundservice@svenskaspel.se.

Figure 6.10 PUL requirements changed


Ditt spel på Keno är betalat!

Välj här om du vill ha e-post efter denna och kommande dragningar - med länk till spännande rättning av ditt Kenospel.

Skicka e-post med rättning. På

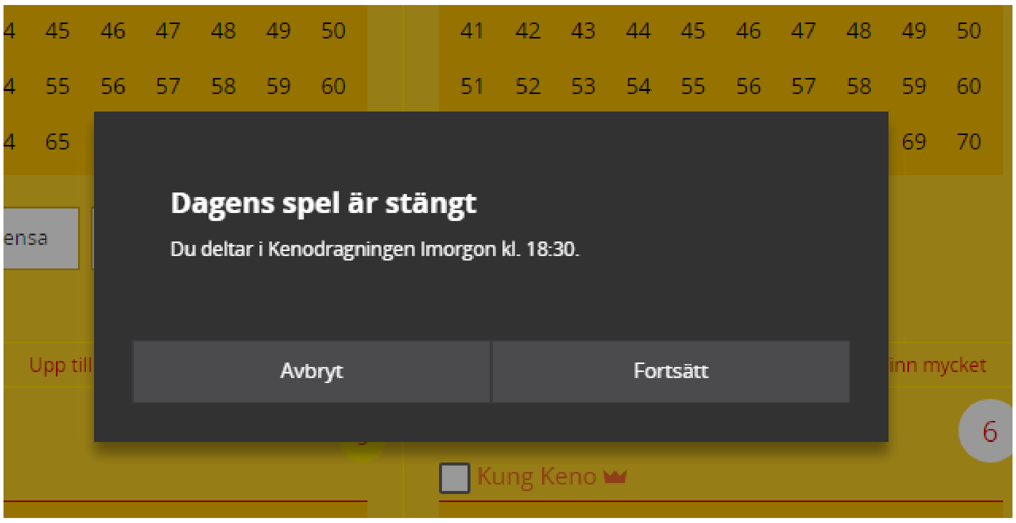
E-postmeddelandet skickas till: brottarn13@testb.svenskaspel.se.

Avbeställ e-post för rättning under Kommunikationsval. Ändra din e-postadress i Min profil.

E-post med rättning skickas inte för bolagsspel, andelsspel eller systemspel.

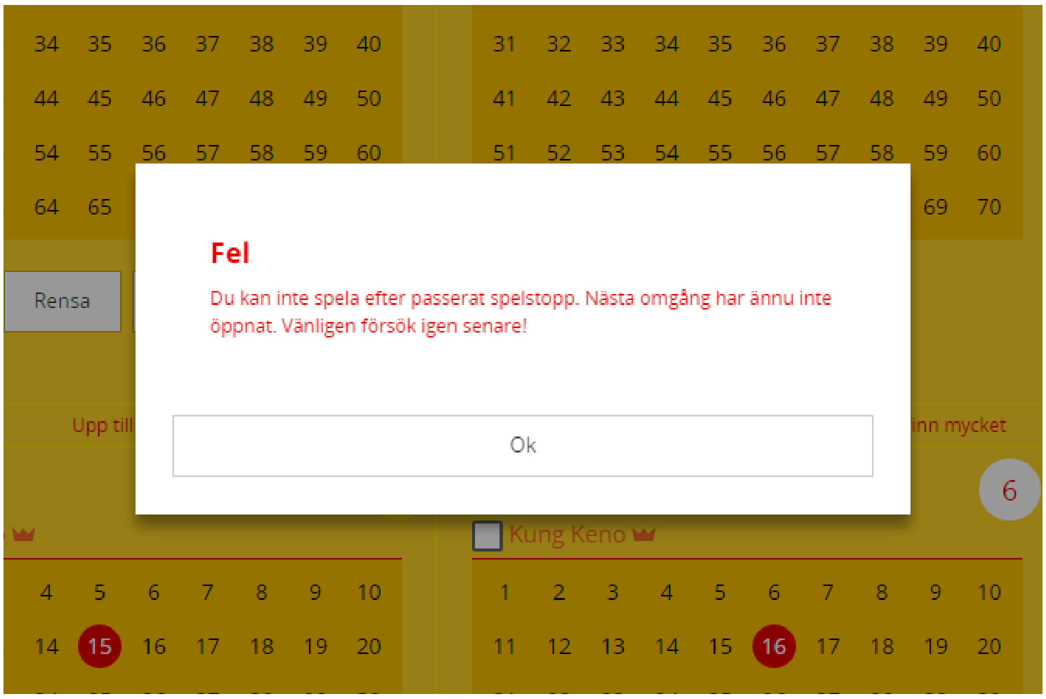
Du har ingen registrerad e-postadress. En registrerad och bekräftad e-postadress är nödvändig för att få e-post med rättning.

Figure 6.11 Result mail popup


Dagens spel är stängt

Du deltar i Kenodragningen Imorgon kl. 18:30.

Figure 6.12 Keno game closed



Fel

Du kan inte spela efter passerat spelstopp. Nästa omgång har ännu inte öppnat. Vänligen försök igen senare!

Figure 6.13 Keno game closed while placing bet

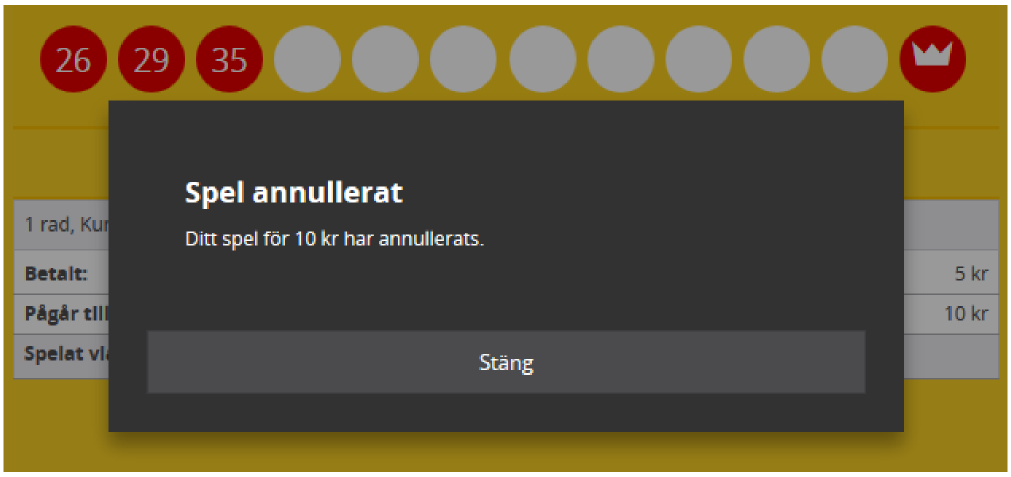
Two measures were applied to limit the effects of unconsidered popups. Prior to starting the test run all users are validated and all flags related to popups are cleared. The timing of when a test run is executed is also implemented to avoid games closed popups.

The main issues that caused flaky test results were:

1. Elements were hidden underneath other objects and thus could not be interacted with.
2. Unexpected popups.

After applying corrective actions to these two issues test cases are executed successfully with a 100% success rate, thus answering RQ2. This is also partly in line with the research done by Hammoudi et al [17] where one of the issues that broke test cases where Popup boxes (4,98%) caused broken test cases.

Vikinglotto 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 (figure 6.14, code 6.15 and code 6.16), as the probability of name changes is less than components being relocated on the web page.



Spel annullerat

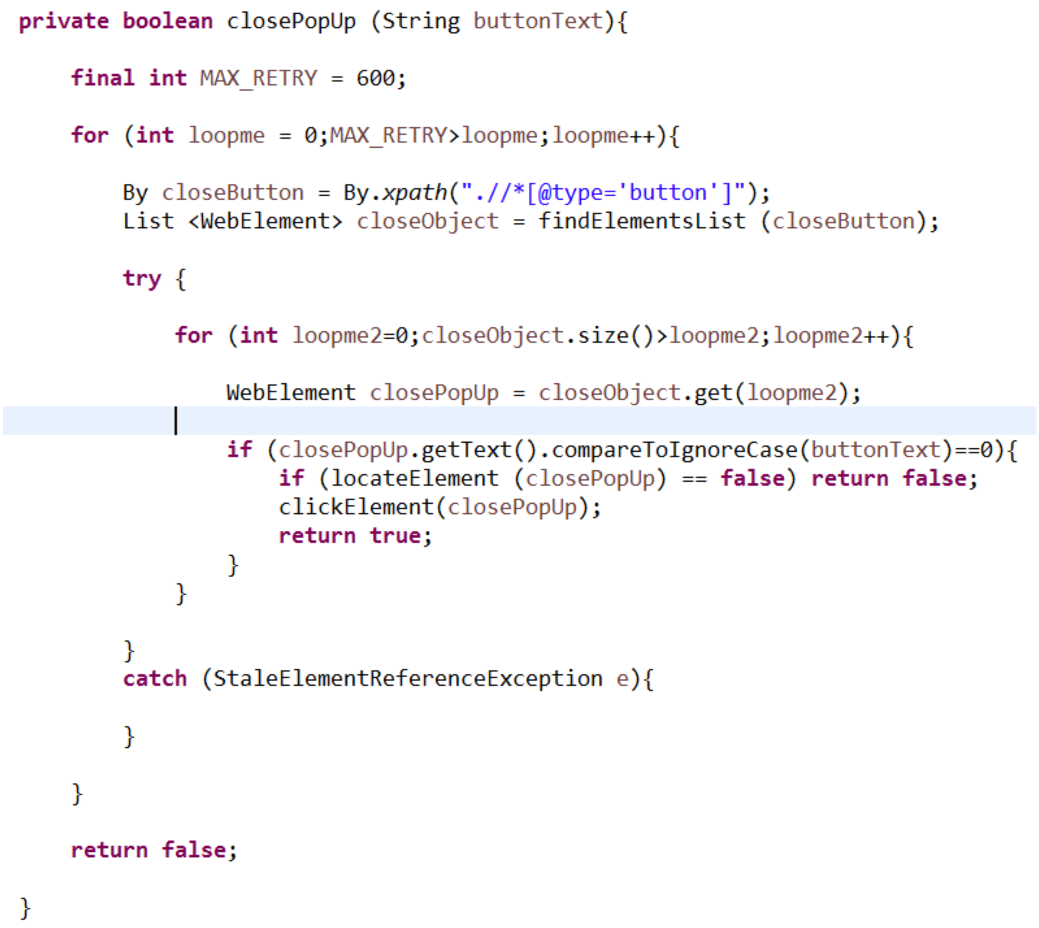
Ditt spel för 10 kr har annullerats.

Stäng

Figure 6.14 Close button

```
public boolean selectAnnulleraClose(){

    return closePopUp ("Stäng");

}
```

Code 6.15 selectAnnulleraClose method



```java
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;

}
```

Code 6.16 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.

<Add a reference to the XPATH report rapporten finns utskriven hemma kolla och fixa>

7 Conclusion

<Add a reference to the Engstöm om att testning ska studeras som en helhet rapporten finns utskriven hemma kolla och fixa>

From the research the conclusion is that regression testing needs to be studied in a wider scope. Decisions made already in the design phase affects the later stages of the project where regression testing is one of the phases that will affect the product during its complete life cycle.

During software development the characteristics of the software is monitored with its set of parameters whit CPU load, Memory consumption, Response times etc. This research shows the needs to also monitor the test execution times as a part of the software characteristics. Not from a user affecting perspective but from a software lifecycle cost perspective, and design decision also needs to consider how testing as a whole is conducted to reduce costs of regression testing.

It is difficult to compare web applications due to the diversity of the functionality they implement. However when it comes to testing they share the same common problems of ensuring that web elements can be interacted with in a reliable way. This finding is applicable to all web applications. The need for this also increases for web applications that are optimized for a smaller mobile device screen size. In these case scrolling elements into view increases significantly.

The scope and time constraints did not allow for a wider set of products to be tested on. Also most of the components are released and hence can’t be modified for testing purposes. Also due to parallel testing with other teams resulted in the product selected to be unusable for longer time periods. If a wider range of products being modified would have been used RQ3 could have been studied in a better way. Also the choice of using only one browser in the beginning showed to be a bad decision. A plan B should have been implemented sooner, his would have given a better test result in the field of reliability of the test results (RQ2).

7.1 Future Research

The method used to locate a web element can significantly affect the execution times. Preliminary research shows that using individual ID locators to find objects are considerably faster. However a wider study where the different options on how the element is found and scrolled into view is not done. Research in this area could further improve the execution times when the GUI is used.

A second research area which is already mentioned is to look into the impact of taking regression testing and testing needs in general into account already when code is developed. And how that would impact the cost of regression testing over the entire lifecycle of the product.

References

[1] MSDN Magazine (2013 November issue) ASP.NET – Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[2] Wikipedia (2017) Single-Page application. [Online].Available: https://en.wikipedia.org/wiki/Single-page_application

[3] ISTQB Glossary regression testing. [Online]. Available: http://glossary.istqb.org/search/regression%20testing

[4] Chittimalli, P.K. and Harrold, M. (2009). Recomputing coverage information to assist regression testing. IEEE Transactions on Software Engineering, 35(4) p.452-469

[5] IEEE STD 610.12 (1990) IEEE Standard Glossary of Software Engineering Terminology. [Online]. Available: http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf

[6] Wikipedia (2017) Document Object Model. [Online] Available: https://en.wikipedia.org/wiki/Document_Object_Model

[7] World Wide Web Consortium (W3C) (2017) What is the Document Object Model. [Online]. Available: https://www.w3.org/TR/WD-DOM/introduction.html

[8] Selenium HQ (2017) [Online]. Available: http://www.seleniumhq.org/

[9] A. Zarrad, "A Systematic Review on Regression Testing for Web-Based Applications" *Journal of Software*, vol. 10 no. 8, p. 971–990, 2015.

[10] Apache JMeter (2017) [Online]. Availabel: http://jmeter.apache.org/

[11] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Automated generation of visual web tests from DOM-based web tests" in *Proceedings of the 30th Annual ACM Symposium on Applied Computing 2015 p.775-782*

[12] M. Leotta, D. Clerissi, F. Ricca and P. Tonella "Capture-Replay vs. Programmable Web Testing: An Empirical Assessmen during Test Case Evolution" in *20th Working conference on Reverse Engineering (WCRE), 2013 p.272-281*

[13] K. V. Aiya and H.Verma "Keyword driven automated testing framework for web application" at 2014 9th International Conference on Industrial and Information Systems (ICIIS)

[14] M. Leotta, D. Clerissi, F. Ricca and C. Spadaro "Improving test Suites Maintainability with the Page Object Pattern: An Industrial Case Sturdy" in *6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2013 p.108-113*

[15] M. Leotta, D. Clerissi, F. Ricca and C. Spadaro "Improving test Suites Maintainability with the Page Object Pattern: An Industrial Case Sturdy" in *6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2013 p.108-113*

[16] M. Leotta, D. Clerissi, F. Ricca and C. Spadaro "Improving test Suites Maintainability with the Page Object Pattern: An Industrial Case Sturdy" in *6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2013 p.108-113*

[17] M. Hammoudi, G. Rothermel and P. Tonella "Why do Record/Replay Tests of Web Applications Break?" in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST) th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2016 p.180-190*

[18] A. Stocco, M. Leotta, F. Ricca, P. Tonella "Why creating Web Page Objects Manually If It Can Be Done Automatically?" in *10 international Workshop on Automation of Software Test (AST) 2015 p.70-74*

[19] XPATH fixa ref

[20] ENGSTRÖM fixa ref

A Appendix 1

KVAR ATT GÖRA:

- Fixa några länkar
- ~~Kolla alla tabeller, figurer~~
- ~~Fix Grafer Tabeller osv I kap. 4~~
- ~~Fixa pratet I abstract~~
- Backup på all data
- Läs ignom kap. Och tänk på imperfekt / presens blanda inte om möjligt
- Stavning
- Video med en test körning
- Powerpoint 7 slides
- Tänkta opponent frågor
- Skriv ut alla referenser och indexera dessa
- ~~Slutlig layout och sidbrytningar~~
- ~~Backup på bilder och UML, koden~~
- ~~Uppdatera tab över kap nuffrorna~~
- ~~Byt Web -> Selenium~~
- ~~Fixa samma ordning Keno, Vikinglotto, Eurojackpot~~

Från Johan

- ~~Tabeller och figerer numrerade enligt huvud kapitlet~~
- ~~Varje huvud kapitel börjar på en ny sida~~
- ~~Inte stapla rubriker alltid text under en rubrik ex. 4 4,1 och 5.1 5.1.1~~
- ~~Tydligare grafer och figurer~~
- ~~All text är inte höger-vänster justerad~~
- ~~Figure 4.2.5 och framåt: tanken är att man här ska kunna jämföra API med Chrome, men jämförelsen mellan graferna blir missvisande eftersom samma test har olika färger för API och Chrome (Logout är grönt på API och mörkblått på Chrome) samt att skalorna är olika (0-1200 på API och 0-18000 på Chrome). Om ingen jämförelse ska göras är det i så fall bättre att placera dem under varandra eller med mellanrum mellan graferna så det är tydligt att de ska ses som separata enheter.~~