DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Making test automation sharable:

The design of a generic test automation framework for web based applications

ANNIKA STRÅLFORS

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Making test automation sharable: the design of a generic test automation framework for web based applications

Annika Strålfors
Royal Insitute of Technology
Stockholm, Sweden
stralf@kth.se

ABSTRACT

The validation approach for assuring quality of software does often include the conduction of tests. Software testing includes a wide range of methodology depending on the system level and the component under test. Graphical user interface (GUI) testing consist of high level tests that assert that functions and design element in user interfaces work as expected. The research conducted in this paper focused on GUI testing of web based applications and the movement towards automated testing within the software industry. The question which formed the basis for the study was the following:

How should a generic test automation framework be designed in order to allow maintenance between developers and non-developers?

The study was conducted on a Swedish consultant company that provides e-commerce web solutions. A work strategy approach for automated testing was identified and an automation framework prototype was produced. The framework was evaluated through a pilot study where testers participated through the creation of a test suite for a specific GUI testing area. Time estimations were collected as well as qualitative measurements through a follow up survey.

This paper presents a work strategy proposal for automated tests together with description of the framework system design. The results are presented with a subsequent discussion about the benefits and complexity of creating and introducing automated tests within large scale systems. Future work suggestions are also addressed together with accountancy of the frameworks usefulness for other testing areas besides GUI testing.

Keywords

Test automation, Automation framework, BDD, Regression testing, GUI testing, Browser automation

1. INTRODUCTION

Software testing is a rapidly growing industry that aim to increase control and reduce risks within large scale applications.[2] The concept of software testing is defined as "*the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior*". [3]. A subset of software testing includes interface testing. Interface testing is performed to evaluate whether systems or components pass data and interact correctly to one another [4].

This study focused on GUI testing and the process of testing web user interfaces. A web user interface constitutes of "*all aspects of a website or web application related to content, functionality, navigation, interaction and presentation that are relevant for using a website or web application*"[5]. Testing graphical user interfaces, GUI testing, is a standard validation strategy for interactive software. In order to perform GUI tests on web applications, test designers write test cases with the purpose of covering different interaction flows. As a system grows, previously tested software is often changed or interfaced with new functionality. This requires the need to verify that previously tested software still work as expected. In software testing, the process of verifying that previously tested software still meets the requirements is called regression testing. Expanding domain sizes challenges test designers to increase the extent of test cases needed for regression tests, which can demand high business costs and time investment (Figure 1).

A common approach for executing GUI tests is to manually execute them on the interface [1]. Manual testing is a testing process that is defined by a test or collection of tests that requires an operator [6]. When performing manual regression tests on a web GUI, testers mimic end user scenarios by interacting with the web application and verifying the software behavior. For repetitive test tasks, such as regression testing (See Section 2.1), the work load for performing manual GUI tests will increase as the web application expand and additionally cause higher risks of bugs remaining undetected due to human risk factors or lack of testing time.

As a consequence of the high business costs and the high quality risks that can come when manually testing large scale applications, a need for automated regression tests have evolved. In contrast to manual GUI tests, maintaining and expanding an automated testing framework requires a level of continuous development. If testers do not possess the development skills required for maintaining the framework alone, the need for adding development skills as a part of the testing resources can become necessary. However, in complex web applications with large domain sizes, multiple browser support and division between developers and testers, performing quality assurance on software through automated GUI testing can be an extensive challenge.

1.1 About the company

The study was carried out at an omni-channel e-commerce consultant company with a broad range of e-commerce solutions. The company provides development and integration of different channels for the client's customer's shopping experience, including online stores, mobile app stores and retail stores.

Although the clients web applications differ in their product market and graphical design, they have a set of shared functional features due to their common market of e-commerce. For example, all web sites include a basket component, a product search functionality and a way of placing an order. Some websites include users to register accounts, while others are designed for a specific customer target. The company uses an agile development strategy where consultants work in sprints. Before an upcoming release they do manually regression tests in order to make sure that the software works accordingly to the requirement specifications from the client.

With the company's solution base consisting of multiple websites, multiple system environments and a variety of omni-channel platforms and browser support, manually regression tests take up a lot of resources. Some of the projects have testers as part of their team, while other include business client resources in the testing processes. There is a variety in development skills among testers were most do not possess any programming skills. For one of the projects, the estimation for manual regression tests before the last release was estimated to 120 hours. As in any dynamic context, the estimations do not always match the subsequent work. Sometimes test time exceeds estimations due to undetected production bugs that adds up the test load, or that there is simply not enough test time left prior to a release.

There was an expressed desire of changing the companies test culture and start using automated tests. These issues derived from the interest for investigating the possibilities of having a generic test automation framework and examine how to adapt the work practices at the company in order to integrate automated testing at the company.

The following section describes the requirement on the automation framework that was collected after a meeting between project managers at the company.

1.2 Goals and requirements

The company wanted to investigate the possibility of creating a generic automation framework that could be shared and expanded between projects. A reason for the generic aspect was to introduce a shared test platform and thereby make it easier for projects to start implementing automated tests. Another aspect for the generic strive was the range of shared software features that advantageously could be utilized within test automation.

1.2.1 Requirements

The following requirement for the automation framework was specified:

- *Test cases should be written by testers (no demand of programming skills)*
- *The framework should be based on open source complementary frameworks*
- *The framework should have multiple browser support*
- *The framework should be possible to use within all company projects*
- *The framework should include standard actions that are shared between projects*
- *The framework should include the possibility of running tests parallel on different browsers*

2. THEORY & RELATED RESEARCH

In the following section theory related research within test automation is addressed together with key concept descriptions.

Testing should ultimately lead to lower business costs, but having an inefficient testing strategy can lead to high economic impact. Studies within the field have estimated that testing can consume fifty or more percent of the total development cost [7]. Other studies suggest that testing approximately accounts for up to sixty percent of the development costs where fifty percent comes from regression testing [8]. Figure 1 shows a general visualization of the differences between manual testing and automated testing when it comes to test coverage and time and cost investment.

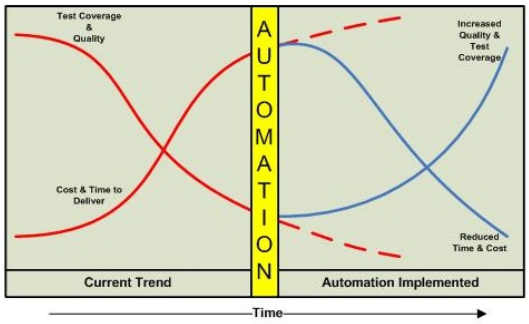

Figure 1: ***Comparison between manual testing and automated tests. Manual testing have low costs and high test coverage if the system is small, while the test coverage decreases and the time to deliver becomes higher as the system grows in size. Test automation requires high costs and low test coverage in the beginning, but as more test cases are implemented and used, the test coverage increases while demanding lower time and cost investment.[9]***

2.1 Regression testing

Regression testing is defined as "*testing required to determine that a change to a system component has not adversely affected functionality, reliability, or performance, and has not introduced additional defects*"[10] . Regression tests are

thereby performed in order to verify that previous developed features still work as expected after software changes. Its repetitive nature has made regression testing a popular topic for automation.

2.2 Automation frameworks

Test automation includes element and practices from both software testing and software development [2]. A test automation framework provides the basis of test automation by defining an abstract infrastructure through a set of concepts, processes, procedures and environment in which automated tests can be executed. It includes the test creation and test implementation by defining the logical interaction between the components. A great benefit of using an automated test framework is that it can be designed to interlace different team members involved in the development, instead of having a distinct separation of responsibilities between developers and testing specialists. Combining involvement from different groups can make the quality assurance more effective [7].

Depending on how the automation framework separates between test scripts, test data and how locators are designed, different level of abstraction is required. The level of abstraction reflects the need for different levels of programming skills in the team that develop and maintain the framework [2]. In order to allow maintenance for non-developers it is necessary to use a high level of generalization to separate the logic from the test data [7].

2.2.1 Browser automation

Creating a test automation framework for web applications demands the need for automating browser behavior. There are different available tools that provide browser automation by accessing web pages HTML files in order to manipulate interface components and thereby mimic real user interaction. The Selenium Webdriver API is an open source framework for web browser automation that is widely used and is regarded as the most comprehensive web test automation tool [11]. By accessing HTML elements, it it possible to assert if a element it visible on the web page and contain the expected content. Selenium WebDriver supports automation for all common browsers and includes functionality of running test suites parallel to using the Selenium Grid API.

2.3 Test automation approaches

As in any growing industry, there have been a lot of trends within test automation. There are tools that record interaction with browsers in which the tester can assert different software behaviors and replay the recording as a test [12]. This automation approach demands for no programming skills, but have a lot of disadvantages when it comes to maintainability, code re-usability and consistency. The lack of scalability within this approach makes maintaining test cases very hard and unsuitable for a big numbers of tests. Another strategy is data-driven test automation frameworks where the main approach is to distinguish the logic from the test data and store the data in external files. A more popular approach that is derived from the data-driven model is keyboard-driven test automation frameworks. In this approach, data are separated from the logic but the external files does also include interaction commands in the form of keywords (i.e *click*, *submit* or *type*) that is specified together with the test data. These commands are interpreted together with the data and processed within the framework. This approach makes it easy for non-developers to write new test cases and to maintain the tests by updating the external files.

2.3.1 Behavior driven development

Behavior-driven-development (BDD) is a software development strategy that has evolved from test driven development (TDD) [13]. In TDD, test is written prior to development in order to make sure that the solution fulfills the requirements. Since developers often experience a hard time writing, naming and understanding what to test for beforehand, the TDD approach gave rise to behavior-driven-development. Instead of focusing on what test cases to write for a feature, the focus within BDD is instead of writing test cases that reflects the behavior which the feature should imply.

In the BDD approach, a domain specific language (DSL) using natural language constructs form the basis of expressing test cases through user scenarios. User scenarios are often used in the process of documenting the software requirements and performing user acceptance tests [14]. The scenarios are sometimes converted into storyboards that visualize the structure and the behavioral aspects. In the process of converting user scenarios to storyboards, research within the field have addressed a risk of ending up with inconsistency between the different requirement representations. By using the user scenario descriptions when creating executable test cases though BDD, the tests can be directly aligned with the requirements and therefore reduce the risk of omitting or forgetting important test cases.

2.3.2 Cucumber

A popular BDD framework for test automation using natural language for writing test cases is the java based tool Cucumber [12]. Each test consist of a *scenario* and is written in the programming language Gherkin [17]. Each row in a scenario, a *test step*, consist of a keyword followed by a sentence. Each test step should be mapped to a function, called a *step definition*, that executes the desired behavior. The keywords consist of either preconditions to the test (expressed as *Given*), triggers to the system (expressed as *When*) or as post conditions that assert system behavior (expressed as *Then*). Figure 2 show a basic test scenario that consist of a three test steps where each step are mapped to a corresponding function with the task to handle to logic for executing the behavior described in the test step.

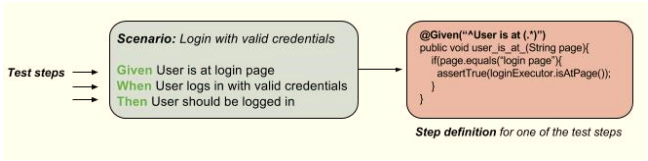

Figure 2: ***Example of a BDD test case. Each test step are mapped to a step definition that handles the logic of executing the test step.***

Test suites can be organized through use of different tags and scenarios can contain tables that pass test data efficiently and run the same scenario multiple times using different data. Every scenario is added in a *feature file*. The feature files consist of scenarios that aim to test a shared component, i.e test that validate user login functionality should be added together in a login feature file.

Case studies that have been conducted on systems that uses Cucumber have showed to improve communication between the team members and deliver test suites on time [14]. However, as the test suites grew in size, a problem emerged when people expressed similar test steps in multiple ways, leading to multiple step definitions that performed the same task. Furthermore, when many people are involved in creating test cases, the study conclude that there is a risk of people using different levels of abstraction in the tests. The main findings from this case study was that in order to make the framework maintainable and consistent, there is a need to select a language owner that can ensure that the test scenarios follow a suitable sentence structure. It is also highly beneficial to use a documentation over the available test steps in order to effectively generate new tests. The study also address that the framework should be developed and maintained continuously, and with this comes a risk factor that the framework can be regarded as time competitive with software delivery.

3. RESEARCH QUESTION

As previously stated, adapting test automation within systems demand for an initial investment as well as a continuous maintenance of the test data. It also often requires a level of development, which can demand the need of adjusting the test resources. It is from this multilayered area the study in this paper originates in order to propose a solution for the following question:

How should a generic test automation framework be designed in order to allow maintenance between developers and non-developers?

4. METHOD

The methodology used in this study is divided into three sections. The first section describes the background research in order to get knowledge about test practices and available resources at the company. The second section describes the process of using the conditions in order to identify a test automation work strategy and propose a suitable system design. The last section address the evaluation of the work strategy and the framework solution proposed.

4.1 Exploratory research

The initial investigation included meetings with the testers at the company in order to get an overview of their current test work and to get their perspective and input on the design components of the framework. In the project with the company's biggest client, a regression testing matrix describing the website features were used as the underlying material for performing manual regression tests. Based upon this material, a document listing potential system functionality (Figure 3) was distributed to all project managers at the company. Their task consisted of going through the list of functionality and mark all the features that their clients web solution contained. The finding of this investigation formed the basis of mapping the shared website features.

FEATURE: REGISTRATION	ELKJOP	STADIUM	ALKO	AXFOOD	MARTIN & SERVERA
Scenario: Registration of a New Private Customer					
Scenario:Registration of a New Business Customer					
Scenario: Unsuccessful Registration of an Existing Customer					
Scenario: Error Msg Displays if Required Registration Information is Missing					
FEATURE: LOGIN/LOGOUT	**ELKJOP**	**STADIUM**	**ALKO**	**AXFOOD**	**MARTIN & SERVERA**
Scenario: Successful Private Login with Valid Credentials					
Scenario: Successful Business Login with Valid Credentials					
Scenario: Unsuccessful Private Login with Unvalid Credentials					
Scenario: Unsuccessful Business Login with Unvalid Credentials					
Scenario: Successful Social Login with Valid Credentials					
Scenario: Log Out					
FEATURE: PRODUCT SEARCH	**ELKJOP**	**STADIUM**	**ALKO**	**AXFOOD**	**MARTIN & SERVERA**
Scenario: Search for Product by Name					
Scenario: Search for Product by SKU					
Scenario: Search for Product by Brand					
Scenario: Search for Product by SAYT (search as you type)					
Scenario: Search for Product by Category					
Scenario: Error Msg on Search for Unvalid Product					
FEATURE: FILTERING/SORTING	**ELKJOP**	**STADIUM**	**ALKO**	**AXFOOD**	**MARTIN & SERVERA**
Scenario: Filter products by Category					
Scenario: Filter products by Subcategory					

Figure 3: ***Project managers was asked to identify the functionality within their web solution. The green marking implies that their client solution included the functionality, while red marking implied that the client did not have the listed functionality.***

4.2 Design work

As an initial stage of the design work, a brainstorming session was set up with the CTO (Chief Technology Officer) at the company. The requirement of that the testers should not need to have any programming skills in order to write the test cases was discussed, and a work strategy based on behavior-driven-development was chosen due to its use of natural language which made it easy to write and understand the test cases as well suitable to use for sharing the test cases to the clients as part of the requirements work.

A first low-fi prototype paper sketch of the system design was created and presented to the CTO. Input arguments as well as output sources was defined. The next step consisted of creating a proof-of-concept working prototype in which a small set of test cases was implemented for two clients web sites. The test described the following user scenario:

1. *Search for a predefined product*
2. *Add the product to the basket*
3. *Assert that the basket counter have increased by one*

The purpose of the test was to validate a shared function between the sites, with the only difference that one of the websites required the user to pick a product size before adding the product to the basket. One part of the goal was to assert that the implementation could reduce the effort needed to implement the shared test steps after it was

implemented for one of the projects. The other part was to make sure that the additional test step required for completing the tasks (picking a size) could easily be added. The proof-of-concept prototype was demonstrated for the testers and project management. Feedback was gathered and documented through notes.

A more comprehensive test implementation for two of the clients was conducted followed by demonstration for one of the clients. The framework was presented on an internal conference held at the company.

4.3 Evaluation

The framework was evaluated through a pilot study in which two testers wrote test cases for a specific regression test suite. A pilot study can uncover problems through a limited number of users and thereby provide important feedback about the work strategy proposed [15]. The test suite consisted of a administration website feature, called *My pages*, for one client.

The test cases included covering regression testing tasks for the following features within My pages;

- *Account management*
  - Change login information
  - Remove account
- *Customer address information*
  - Update address tests
- *Newsletter subscriptions*
  - Subscribe to newsletter test
  - Unsubscribe to newsletter test
- *Wishlists*
  - Create a wishlist
  - Remove a wishlist
  - Share a wishlist
  - Add products to a wishlist

A total of 24 user scenarios was created through Cucumber syntax. Time estimations for test creations and implementations was collected as well as execution time.

In order to gather qualitative feedback about the new test work strategy, framework usability and future use, a survey was distributed to testers, project managers and the business area manager.

5. RESULTS

This section presents the results of the work strategy proposal, the system design, the pilot study and the evaluation survey. The results are followed by a subsequent discussion.

5.1 Work strategy proposal

In order to replace manual regression testing with test automation it needs to be seen as a part of the development. Testers need to work together with developers to produce efficient test cases. Due to that the testers at the company having much knowledge about customer requirements regarding acceptance criteria, which is often expressed similar to user scenarios, using BDD for writing test cases is proposed. Writing test cases as user scenarios makes them very easy to understand, which makes it possible to involve customers in the test creation process. Using Cucumber did thereby align with the companies requirement about that the test cases should be able to be written by testers without programming skills, as well as be based on complementary open source frameworks.

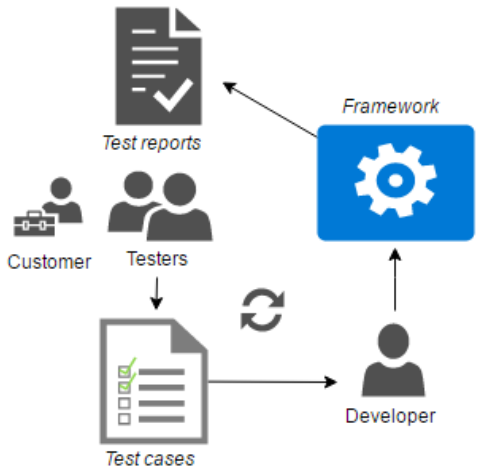

Figure 4: ***Work strategy proposal. Testers write the test cases based on requirements from the client, where one of the testers is regarded as the language owner. The test cases are showed and discussed with a developer that implements the test case within the framework. The tester executes the tests and analyse the automatically generated test reports.***

The proposed work strategy are visualized in Figure 4 and consists of testers writing the test cases, where one tester in each project should be regarded as a *language owner*. The language owner insures that the test cases follow proper structures and that there are not any duplications of step definitions. The language owner should hence check all new test cases and assert that they follow proper syntax and reuse previously implemented step definitions if possible.

New test cases should be presented and discussed with a developer that implements them within the framework. For new development, developers will see what GUI elements will be included in the tests and add proper locators (preferable by setting id tags on the HTML element under test). The tests suites should be set up to run automatically on defined events (i.e after a deploy) and testers should validate the results through generated test reports. Bugs within systems are distributed to developers through existing company internal systems.

5.2 Luna design

The automated framework was named Luna. As stated by company requirements, the framework should use additional open source frameworks. The technology for the automation framework design were decided to be the Selenium Webdriver for browser automation and Cucumber framework for test creation. The framework was written in Java. Applying Luna on a web application includes creating a site specific project that extends a shared common project which constitutes of the generic layer of the framework. A description of the system design components will be addressed below.

5.2.1 Page object design pattern

The framework uses a page-object pattern design. Page-object design pattern is a popular test design approach within automated testing [16]. A page object is an object-oriented class that serves as an interface of the web application under test. A page object class holds the page element locators that are related to the specific page. Within GUI testing of web applications, the locators consist of variables that are used by Selenium Webdriver in order to access specific HTML page elements through use of id, xpath expressions, css or class names. Furthermore, page object classes handle the logic for interacting with page elements. In the proposed framework design, a page object pattern design is used to apply a single responsible principle for the classes by isolating the logic to a specific web component or web page view. A page object class covers the logic for either a website page view or website component. For example, the GUI of the product page is regarded as a page while the website header is regarded as a component. The product page class holds the locators for the HTML elements included in the product page GUI, as well as including functions for performing user interaction on the product page, i.e clicking on the buy button or adding the product to a wish list.

5.2.2 Executors and action packages

The framework separates the test logic in two main packages; actions and executors. For a test step, action classes hold the corresponding step definition. Action classes handle assertions and trigger GUI interaction by calling proper executors. The executor handles the logic of executing the action through browser automation. Besides containing functions and locators, the executors also contain an additional locator, a variable called *page locator*, for asserting that the specific page object is visible at a certain event. For example, if a test includes asserting that a user is directed to a specific page after a specific user interaction, the element visibility of the page locator is used to assert that the user was in fact directed properly. Figure 5 shows an example of action classes from the framework and their corresponding executor classes.

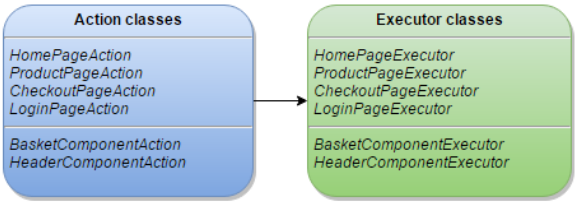

Figure 5: ***Design concept of the framework: Action and executor classes. Action classes consist of the functions that the Cucumber test steps are mapped to. The action classes calls the proper executors in order to execute the test step on the browser through Selenium WebDriver.***

5.2.3 Framework layers

At the brainstorming session it was decided to move forward with an architectural design of dividing the framework into two main layers; a common layer and a customer layer. The common layer consists of the generic logic which includes initialization of the automation drivers, shared action classes, shared executor classes and framework configuration. The customer layer consists of test cases, site specific actions and site specific executors. The two framework layers are visualized in Figure 6. Each project that uses the framework will need to add a client specific project in which they add action classes for that client's specific step definitions and all executor classes where the locators should be initialized. The client specific executor classes extend the common executor classes and send the locators to the common layer through the constructor.

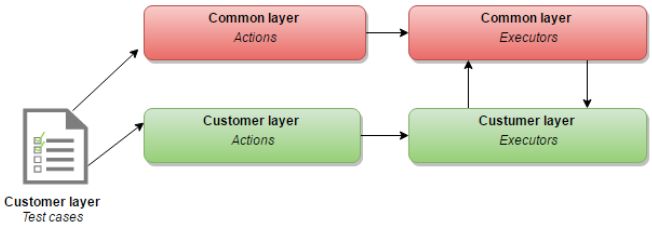

Figure 6: ***Design concept of the framework: A common and a site specific layer***

5.2.4 Running the framework

The framework has three input arguments; a url, a browser choice and a variable for determining if it should run on a desktop or mobile design. Testers executes the test through the console window where the url, browser and size are passed in as input arguments. The supported browsers are Chrome, Firefox and Internet Explorer. For each test step, the framework will go to the corresponding action (available in the common layer if it is a shared test step or otherwise in the customer specific layer), the action class will call its executor class (in the common or customer specific layer) that performs the action on the browser. The testers have the possibility to see the test execution since the framework will open a browser window in the execution. In the console, the testers can see what test step that are currently executed and follow the execution live.

5.2.5 *Test report system*

After each test run, a report file is generated (Figure 7 and Figure 8). For test report generation, an open source cucumber plug-in was integrated with the framework. The reports include test execution documentation through error logs, print screens of test failures, time statistics and test report visualizations. If a test step fails, it is marked red together with error log information. The developer can set up error log messages for each test step in order to make it easy to follow up bugs.

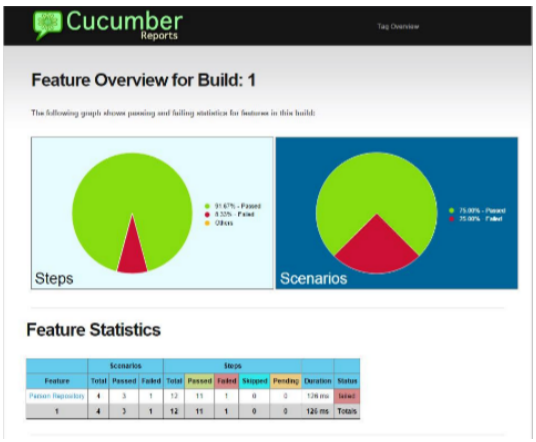

Figure 7: ***Cucumber test report plugin: Visualizations over test executions. The tester can see how many test cases passed the execution together with time reports.***

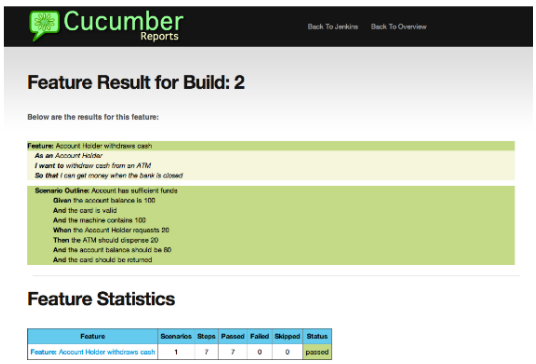

Figure 8: ***Cucumber test report plug-in: Coloring of test step execution results. The tester can look at every test scenario in order to see the results of every test step. Succeeded test steps are colored green while failed test steps are colored read together with error logs messages.***

5.3 Evaluation

5.3.1 *Pilot study*

Task	Hours
Writing test scenarios	22
Implementing test scenarios	40

Table 1: Time estimations from the pilot study for writing and implementing the tests.

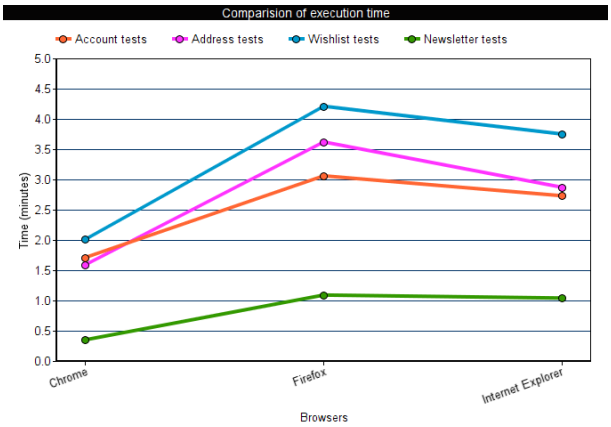

Figure 9: ***Execution time for My pages regression test suite between different browsers. The test features contained of 6 account scenarios, 7 address scenarios, 2 newsletter scenarios and 9 wish list scenarios.***

The pilot study consisted of two testers that used the suggested BDD approach and wrote 24 user scenarios for a particular regression test suite, My pages. Initial meetings were held in order to introduce behavior-driven-development and gather input from the testers. The testers received a list over the currently implemented test steps within the framework. There were initially some distinctions between how the testers expressed test cases using natural-language constructs. The need of having a user friendly documentation over the current test steps as well as a shared template for how to express user scenarios were raised. Table 1 describes the test estimations for writing and implementing the test cases in the pilot study. Figure 9 show the execution time for the test cases for different browsers.

5.3.2 *Survey results*

The survey was distributed to 5 participants at the company. They were the two testers that had been involved in writing the test cases for the pilot study, the two project managers for the two clients that the pilot study was conducted to, and the business area manager at the company. On the question *Do you believe in a shared platform for automated testing within the company?*, all respondents answered that they did. On a questions about the obstacles and risks of adapting automated tests within a project, the testers expressed a concern for the time amount needed to implement the tests and a risk that the company rely too much on test results. The main benefit of adapting automated testing was perceived to be a shorter regression testing time, faster covering of bugs, time released to perform more advanced functional and investigative tests. The testers also got a list of different work tasks and was asked to to express their perceived future involvement with Luna. The result are

shown below in Figure 10.

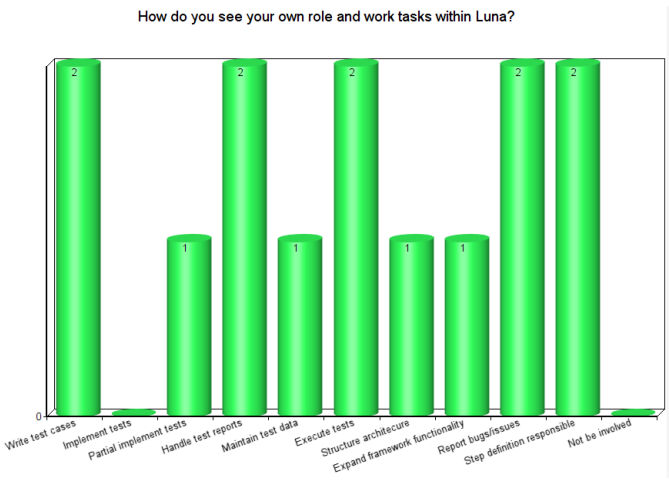

Figure 10: ***Testers perceived involvement with Luna***

The project managers expressed a concern for being unable to change the test work strategies within the team and that the framework would stop being maintained and expanded. The main benefit of using automated tests was perceived to be high development quality, faster feedback to the developers, less time for repetitive work and human risk factors. A benefit of changing the test culture within the company was also found. The project managers expressed that in order for the test automation to work within their project, developers needed to be involved. The need of having BDD test cases as part of the development requirements was also addressed.

The impressions of Luna were positive among both testers, project managers and the business manager. A tester addressed the importance of adapting to Luna in small scale in order to further evaluate its use. One of the project managers expressed that Luna had potential of becoming the company's standard choice for test automation. Both project managers said that their project potentially had resources for adapting the work strategy solution. The customers interest with Luna was regarded as high.

6. DISCUSSION

Results from the pilot study indicated that testers had a positive experience using behavior-driven-development. Since the pilot study was conducted on testers that had not used behavior-driven-development before, the learning process needs to be accounted for when reading the time estimations. The time can therefore act as an indicator but not exclusively be seen as a description of the time effort needed to write tests. It is likely that as testers get used to writing tests cases, the amount of time will decrease.

The implementation time exceeded the time for writing tests, which were expected since the implementation requires more work. Since the tests were implemented for an area that did not yet include automation, the benefits of the generic framework part could not be used. If applying automation tests for an another website containing My page's functionality, it will be possible to reuse test steps and execution logic.

A need that was addressed when discussing the test design was the need of having a documentation over implemented step definitions. An aspect of natural language syntax is that people express processes in different manners, increasing the risks of test step duplication if not handled correctly. This could be seen in the pilot study where the testers initially used different words and sentence structures for expressing test steps. The need of having a 'language owner' was agreed upon, which coincide with findings in previous research studies[14]. As seen from Figure 10, both testers that participated in the pilot study answered that they could agree to be language owner (described as 'Step definition responsible'). The only task in which both testers answered that they did not perceived them self to be suited for was the implementation of the test cases. This aligns with the need to adding developers as part of the test resources. One tester answered that he could be partially involved within the implementation, which is valuable for raising the understanding of the framework as well as relieve the work load for developers.

Moving towards automated testing within a large scale company can be a challenging area. For a consultant company, a crucial aspect for the survival of the framework is that the company manages to motivate its use for the customers so that the customers see the benefits of investing in test automation. The demonstration for the customers throughout the study have shown to be successful. The current stage is awaiting the pilot demo for the involved customer. After the demonstration, the hope is that the customer agrees to proceed with the framework. If proven successful for one customer, it will be easier to sell to other customers and thereby adapting the generic use.

Another challenge is to adapt test automation as a part of the project work. Manual GUI regression testing does not demand the involvement of developers while adapting the work towards test automation demands for the developer's participation. Within the constantly expanding industry of e-commerce, finding resources and time to invest in new areas can be challenging.

This study concludes that behavior-driven-development is a good test design approach when test is to be written by non-developers. Testers have knowledge about requirement specifications, and using requirement specifications as a foundation for test cases can be an effective approach for test automation. Another great benefit of expressing test cases through natural language is that the clients can be involved in asserting that test cases cover the requirements.

Using Selenium Webdriver for browser automation requires an initial learning process. Web browser automation can sometimes be tricky due to page time loads and complex dynamic website logic. Since Selenium Webdriver is a well-known tool for browser automation, there are multiple forums available online. In order to access page elements through using element locators, the need for having good locators within the HTML code is crucial to the time required to implement tests. By involving developers in automated GUI testing this can raise awareness of the importance of developing testable code.

The use of Cucumber together with Selenium WebDriver is exclusively for GUI testing. However, Cucumber can be integrated to work with low-fi testing as well, such as unit tests. In this regard, the functions that the test steps are mapped to should execute unit tests of the code instead of browser automation. Luna framework does not support unit testing, but expressing test cases through BDD could be useful for unit testing as well. For a company to increase the development control and test coverage, unit test should also be used and combined with higher level of user scenario testing through GUI testing. GUI testing can detect all bugs and behavior within the software that can be asserted through the HTML, but it does not necessary give the insights about what caused the wrong behavior. Unit testing that directly test on code level gives more insight about specific program behavior and should therefore be used in combination with GUI tests.

Luna is not limited to only work for e-commerce web solutions. It can be used for GUI testing on any web application. However, the common part of the framework is based on that the web applications shares some common functionality. If Luna is used on web applications that are divese, there would be less use of the generic part of the framework and the reusability of test steps would be more limited.

To cover user flows through regression test automation that replace manual testing demands a specification of the level of test ambition. When manual tests are performed through a human operator, bugs might be detected that were not particularly tested for. The amount of user scenarios in large scale applications can be very extensive, potentially causing a need to prioritizing the automated testing efforts.

The test automation framework could be automatically executed through integration with other tools. Jenkins is a popular tool for continuous integration in which test suites can be set up to run automatically, i.e. after each development deploy. The framework will be integrated with continuous integration and also include version control handling of the common layer of the framework. The requirement of test parallel execution was not fulfilled during the time of this study, but will be added in continued work.

7. CONCLUSION

In this study a system design proposal of a generic testing framework for web applications was created and evaluated through a pilot study and a follow up survey. The work consisted of close collaboration with testers and project managers at the company. The framework was demonstrated for relevant company clients as well as presented internally at a company conference. The overall perception of the possibilities of adapting a generic automation framework within the organization was agreed upon to be an appropriate strategy. A move towards automated tests for repetitive and/or shared test tasks is highly beneficial for large scale companies. The design approach of adapting behavior-driven-development as test design showed to be suitable for the company as it enabled non-developers to produce and distribute test cases within the framework. The similarities between requirement specification, documentation and user scenarios can especially be regarded as beneficial for a consultant company. The possibility of clients to understand the test cases adds value to the working process. Test automation can be challenging and demand changes within a companies test culture and test resources. In order for the automation framework to be fully adapted, clients need to agree on investing in the framework. Furthermore, test processes need to be adjusted to consist of collaboration between test designers and developers. In order to fully evaluate the business value of adapting the proposed framework, further investigation after a period of including automated tests as part of the work routines is needed.

8. ACKNOWLEDGMENTS

Thanks to the company for the possibility of conducting the study together with testers and project managers. I also want to thank my supervisor at the Royal Institute of Technology, Vincent Lewandowski, for following the work process and contributing with valuable feedback.

9. REFERENCES

[1] Tech Target
GUI testing (graphical user interface testing)
http://whatis.techtarget.com/definition/GUI-testing-graphical-user-interface-testing

[2] Sabina Constantinescu, Radu Amaricai
Designing a Software Test Automation Framework
Informatica Economica, p 152-161, 2014

[3] The International Organization for Standardization
ISO/IEC TR 19759:2005, Software Engineering âĂŤ Guide to the Software Engineering Body of Knowledge (SWEBOK).5

[4] The International Organization for Standardization
ISO/IEC/IEEE 24765:2010(en) 3.1489

[5] The International Organization for Standardization
ISO 9241-151:2008(en) 2.7

[6] The International Organization for Standardization
ISO/IEC 13213:1994, 1.4.57

[7] Antonia Bertolino
Software Testing Research : Achievements , Challenges , Dreams Future of Software Engineering, FOSE'07, 2007

[8] Patrick Day
N-Tiered test automation architecture for Agile software systems Procedia Computer Science, p 332-339, 2014

[9] Thom Garrett
Implementing Automated Software Testing - Continuously Track Progress and Adjust Accordingly
http://www.methodsandtools.com/archive/archive.php?id=94

[10] The International Organization for Standardization
*I*SO/IEC 90003:2014(en) 3.11

[11] Dui Wencai, Fei Wang
A Test Automation Framework based on WEB
International Conference on Computer and Information Science, IEEE/ACIS, 2012

[12] Bernhard Hosil, Stefan Sobernig, Mark Strembeck
Natural-language scenario descriptions for testing core language models of domain-specic languages
Modelsward 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, p 356-367, 2014

[13] Niclas Olofsson
Automated testing of a dynamic web application
Linkopings Universitet, 2015

[14] Mark Micallef, Christan Colombo
Lessons learnt from using DSLs for Automated Software Testing PEST Research Lab, Faculty of Information & Communication Technology, 2014

[15] Jonathan Lazar, Jinjuan Heidi Feng, Harry Hochheiser
Research methods in human-computer-interaction
ISBN: 978-0-470-72337-1, 2010

[16] Test Design Considerations
SeleniumHQ web site
http://www.seleniumhq.org/docs/06$_{t}est_{d}esign_{c}onsiderations.jsp$

[17] Arvind Madhavan
Semi Automated User Acceptance Testing using Natural Language Techniques Iowa State University, 2014