Designing a visual regression testing tool

Decrease fear-driven development and enhance the quality assurance

Elin Hörnfeldt

August 21, 2017
Master’s Thesis in Interaction Technology and Design, 30 credits
Supervisor at TFE-UmU: Kalle Prorok
Examiner: Thomas Mejtoft

UMÉÅ UNIVERSITY
DEPARTMENT OF APPLIED PHYSICS AND ELECTRONICS
SE-901 87 UMEÅ
SWEDEN
Abstract

Testing a software is important to maintain the quality of the application. There are many ways of testing functionality of an application but not so many for testing the interface. When reusing CSS is a common approach, one change in the CSS can cause many visual changes on the interface. Not testing these changes, visual errors might occur which can decrease the quality of the application. Also, not knowing where the change is affecting, fear-driven development, i.e. the developers feels fear when changing code, can arise for the developers. In this thesis, a pre-study was made to investigate if any current fear-driven development existed among developers. Then a visual regression testing tool was created, using a rapid prototyping process, to help the developers testing the interface during the evolving process of the application. The tool’s primary purpose is to show images of the visual changes that have occurred for the developers and as a result, the fear-driven development can decrease for the developers and the quality can be improved. The tool was implemented using AngularJS, NodeJS and ResembleJS and was tested on UI developers. The developers got a case where they had made CSS changes and then wanted to see how the changes affected the site they were working on. All of the developers felt, often or sometimes, fear when changing code. After using the tool, their feelings of fear-driven development was decreased and they all saw the tool as helpful when finding visual errors that might occur when CSS or web components are changed.
# Contents

1 Introduction ........................................... 1
   1.1 Swedbank AB ........................................ 2
   1.2 Problem Description ................................. 2
      1.2.1 Cases ......................................... 3
   1.3 Aim and Purpose .................................... 4

2 Theoretical framework ................................ 5
   2.1 Web testing tools ..................................... 5
      2.1.1 Selenium testing ............................... 5
   2.2 Fear-driven development .............................. 5
   2.3 Design process ...................................... 5
      2.3.1 Usability ...................................... 6
      2.3.2 User experience ................................ 6
      2.3.3 From pen and paper to interactive prototype ......... 7
      2.3.4 Designing for developers ....................... 7
   2.4 Visual regression tools .............................. 7
      2.4.1 Existing tools .................................. 7
      2.4.2 BackstopJS ..................................... 7
      2.4.3 Happo .......................................... 9
      2.4.4 Wraith .......................................... 9

3 Tools and frameworks .................................. 11
   3.1 User Experience Questionnaire ....................... 11
   3.2 Interviews .......................................... 12
      3.2.1 Semi-structured ................................ 12
   3.3 Likert scale ......................................... 12

4 Method .................................................. 13
   4.1 Literature Research .................................. 13
   4.2 Overall Prototyping Process ......................... 13
      4.2.1 Rapid Prototyping ............................... 13
4.2.2 User Centered Design .............................................. 13
4.2.3 The Users .......................................................... 14
4.3 Pre-study ................................................................. 14
  4.3.1 Workshop .......................................................... 14
  4.3.2 Questionnaire ...................................................... 14
4.4 Low fidelity prototype ................................................ 14
4.5 High fidelity prototype ............................................... 15
4.6 Final prototype ......................................................... 15
  4.6.1 Implementation .................................................... 15
  4.6.2 Proof of concept .................................................. 15
  4.6.3 Pilot test ............................................................ 16
  4.6.4 Study design ....................................................... 16
  4.6.5 Evaluation .......................................................... 16
4.7 Tools .................................................................... 16

5 Results ................................................................. 19
  5.1 Pre-study ................................................................. 19
    5.1.1 Workshop .......................................................... 19
    5.1.2 Questionnaire ...................................................... 20
  5.2 Low fidelity prototype ............................................... 20
  5.3 High fidelity prototype .............................................. 20
  5.4 Final prototype ....................................................... 20
    5.4.1 System description ................................................ 20
    5.4.2 Usability tests .................................................... 23
    5.4.3 UX questionnaire .................................................. 31

6 Discussion ............................................................ 33
  6.1 Limitations ............................................................ 34
  6.2 Future Work .......................................................... 34

7 Conclusions .......................................................... 35

8 Acknowledgements .................................................... 37

References ............................................................... 39

A Semi-structured interview questions .................................. 43

B UEQ - user experience questionnaire .................................. 45

C Pre-study questionnaire ................................................. 47
# List of Figures

1.1 An example of a diff image. ............................................. 4

2.1 The interface of BackstopJS, no diffs. ............................... 8
2.2 The interface of BackstopJS, with diffs. ............................ 8

3.1 An item from the UEQ. ................................................ 11

4.1 Testing low fidelity. .................................................. 15
4.2 Proof of concept, image differences with a mismatching percentage of 0.02. 16

5.1 Survey picture 1. .................................................... 21
5.2 Survey picture 2. .................................................... 22
5.3 Illustration of the low fidelity prototype. .......................... 23
5.4 High fidelity prototype, the run test view, first iteration. ........ 24
5.5 High fidelity prototype, the diff images view, first iteration ..... 25
5.6 High fidelity prototype, the run test view, second iteration ...... 26
5.7 High fidelity prototype, the diff images view, second iteration ... 27
5.8 Final prototype, the run test view. .................................. 28
5.9 Final prototype, the diff images view. ............................... 29
5.10 UEQ. ................................................................. 31
5.11 UEQ - Benchmark. .................................................. 31

B.1 UEQ. .......................................................... 46

C.1 Questionnaire to the pre-study phase. .............................. 48
List of Tables

1.1 Affects on different pages when changing the CSS of a button. . . . . . . . . 3
Chapter 1

Introduction

Garris Shipon quoted, “Breaking CSS (cascading style sheet) is easy, testing it is hard”, and that is a common matter today [1]. When developing a web application, the functionality, usability, interface, performance and security of the application are important areas that need to be tested to assure good quality and usability of the application [2]. There are challenges though, especially in visual regression testing [3]. Regression testing is the testing process when software is evolving and verified that the modifications of the system had not caused unwanted effects. This is an important step when new changes in a system must maintain the quality [4, 5]. Visual regression testing is the testing of visual changes during the evolution process of the system. It can also be named as a form of CSS testing [3].

More and more companies and developers are starting to realize the value of using a visual regression testing tool. BBC News, a news production company [6], is one group that notices how small CSS or other code changes, appeared release after release and as more members joined the team [7]. Developers are changing CSS, or other code, in a system all the time during the development process of a product, and it is not unusual that the same code is duplicated and used in several places in the application. For example, there are many benefits of using the same CSS rule on multiple web pages: the visual style will be more consistent, the maintainability improved and downloading speed increased [8]. Web components, which can be described as a collection of different technologies that allows the developer combined markups and styles into customized HTML tags [9], is another example of elements that occurs in several places on a website. When a developer rewrites code, in CSS or in a web component, the changes can occur somewhere the developer is not aware of because the code can be used at so many places. Also, the developer has to manually look for the changes and see that it looks like it is supposed to. This takes time and resources from the company and it is a significant risk of missing something out. The developer must be sure that the quality of the website is retained, which is not always clear when the same code is used on many pages. This can also contribute to fear-driven development for the developer, i.e. the developer feels fear of changing code because he or she does not know how the change will affect the whole site [10].

There are some tools and tests that a developer can use to track changes. For example, Selenium¹ and Sahi² are automated web testing tools that are testing the functionality of

¹http://www.seleniumhq.org/
²http://sahipro.com/
Chapter 1. Introduction

the website. There also are some built-in tools in the browsers, (Chrome\textsuperscript{3}, Safari\textsuperscript{4}, Firefox\textsuperscript{5}) that help the developer with debugging and inspecting rendered web pages. A web page will also be rendered different depending on the browser, i.e. the page may not look exactly the same in Safari as in Chrome. Testing the visual changes over different browsers and devices can be done with tools like CrossBrowserTesting\textsuperscript{6} or BrowserStack\textsuperscript{7}. So there are many tools for the developer to test functionality and style issues, but there are not that many tools to use for testing visual regression changes. When a developer makes changes from one version to another, visualizing the changes for the developer can help the developer decrease the feeling of fear-driven development and improve the quality assurance.

This thesis is done in cooperation with Swedbank Group IT, Sweden. They are using Selenium for testing functionality but are in need of a tool that can facilitate the developer’s process of finding visual errors. Swedbank has many developers that are changing code all the time and it is not unusual that the changes are affecting many web pages. But there is a solution to solve the visual regression errors that can occur. By taking screen shots of a page and compare the image to another screen shot from an earlier version of the page, differences in the image can indicate the change. A few variants of this kind of tool have already been made, but for Swedbank a tool that can integrate with their Selenium tests, take screen shots and compare images automatically with a test interface that can show the changes for the developers, is desirable.

1.1 Swedbank AB

Swedbank is one of the largest banks in Sweden and is also leading in the Baltics states with 7.3 million private customers and 650,000 corporate and organizational customer \cite{swedbank}. For Swedbank, the security and quality are essential, and a visual regression testing method that suits their demands could help them improve the quality assurance of their applications.

1.2 Problem Description

It is not uncommon that developers need to do CSS or other code changes that can affect the layout of, for example, a website or a web application. Maybe a button is not precisely placed where it is supposed to be and a little bit of margin is therefore added. This can place the button at the right spot but maybe this button is also used on another page where it will be misplaced after this CSS change. In table 1.1 an example is visualized. The example shows a button on a login page that is placed a little bit too much to the left. By adding some margin there, the problem is solved. Now it looks much better on the login page. But on the home page, were the same button was used, the margin change displaced the button and a visual error occurred.

The Selenium tests are testing functionality. For example, if you click a button and a particular thing then should happen, or if you are filling in a field and a validation text message then should be shown, Selenium verifies that it happens. But, from the previously example, that visual error will not be noticed in the Selenium tests. Selenium tests is a given choice when testing the functionality of a web application over different platforms.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
1 & 2 & 3 \\
\hline
\end{tabular}
\caption{Example of a visual regression error}
\end{table}
1.2. Problem Description

| CSS | .btn {  
|     } | .btn {  
|     margin-left: 20px;  
|     } |

| Home page | ![Home page](image1.png) | ![Home page](image2.png) |
| Login page | ![Login page](image3.png) | ![Login page](image4.png) |

Table 1.1: Affects on different pages when changing the CSS of a button.

and browsers. But there is not any given choice when a developer wants to test visual regressions. On a website with only a few pages, this is probably not a problem when it is easier to find the mistakes with the human eye. But on a site with many more pages, the time and resources are not always enough to find these errors.

One solution to this problem is to take screen shots of all pages and compare with images taken on the pages before the CSS or code change. By comparing the before and after images with each other, any differences can be shown in a diff image. The diff image will be the same image as the two screen shots taken, but with markings on the differences. In figure 1.1 a typical look of a diff image can be seen. In the figure, the ref image is the screen shot taken before the change and the current image is the screen shot taken after the change.

There exist a few tools that take screen shots, compares them and give a diff image as an output. But there is not any optimal tool for automating screen shots with Selenium tests and that also provides a good interface were the diff images can be shown for the developer.

1.2.1 Cases

When is it needed for the developer to get an easy visualization of diff images? Below, some scenarios are stated to better understand when this tool can be helpful.

− When a developer has rewritten a web component but still wants it to look the same as before. Then the goal is that no diff images are generated.

− When a developer makes some changes in a web component that will give any visual change.

− When a developer makes some changes that give visual differences, the developer does not always see if some changes are unwanted. For example, if the developer made a
button bigger, it works fine on most of the pages but on one page the change gave visual errors.

– If the developer is not used to work with UI (user interface) or design, she or he might not be as aware of visual errors as an UI developer might be. Diff images will then highlight the errors for that developer.

1.3 Aim and Purpose

The objective of this thesis is to evaluate how a visual regression testing tool can help the developer to see if visual errors are produced after making changes in the CSS, web components or some other code, and see if the tool can improve the quality assurance on the website and make the developer feel less fear-driven development.

The purpose of this thesis is to create a prototype that can visualize diff images for the developer so he or she can find the errors in the layout and feel less fear when rewriting code and easier see visual errors for improving the quality.

The aim is to make an user-friendly prototype that shows the images in such a way that it is easy to find the diff images that shows diffs the developer are not aware of.
Chapter 2

Theoretical framework

2.1 Web testing tools

To ensure the quality of, for example a website, the developer can use different tests. Testing the software is important to ensure that the program corresponds to its specification and to uncover defects in the software. Testing the software also makes sure that the software is not doing what it is not supposed to do and give the confidence that the system is working correctly. [12].

2.1.1 Selenium testing

Selenium is a free, open source and automated testing framework. With Selenium, the developer can make sure that the application works as it is supposed to do [13, 14]. By using Selenium the developer can, for example, test that if the user pushes a specific button, he or she gets the outcome that is expected.

2.2 Fear-driven development

Fear-driven development occurs when developers feel fear when changing code. Fear of breaking the system when fixing bugs, repair or improve code. This is a common situation, especially in big projects and when new developers enter the projects. The developers might hesitate when issues must be fixed and sometimes they leave the code as it is [15]. Testing the system is important, not just for quality assurance, but also for fear-driven development. Without testing, every code change increases the risk of breaking something. But resource and time limitations make extensive system testings hard to accomplish and therefore the developers often cross their fingers and hopes for the best. Da Veiga [16] propose automatically regression testing to determine if code modifications break current functionality. The regression testing will then result in less fear for the developers.

2.3 Design process

According to Benyon [17], there are a lot of aspects to consider during a design process. To create a prototype that has fulfilled its requirements at the end of the project, some areas need to be considered. First, the problem must be stated and understood. Second, the
designer must figuring out what the system has to do and should be like and work with the
requirements of the product, system or service. Designers needs to investigate the range of
people, activities and contexts for the system so they can understand what is required for
the developing process of it.

Conceptual design is an abstract way to design the system. The design is supposed to
envision the idea for both the designers themselves and also for other people to be able to
evaluate it. There are many tools for this, but all have the purpose of bringing the abstract
ideas to life. Every design is also followed up by an evaluation. An evaluation can take form
in many different ways. It can be just the designer checking through the design to make sure
everything is completed and correct or making it to a scenario for people to work through.

2.3.1 Usability

There are a lot of parts to take into account when designing a system with good usability.
To get a high degree of usability, the system should let people do things with an appropriate
amount of effort, should contain appropriate functions and information content and be
organized in an appropriate manner. It should be easy for the user to learn how to do
things and remember how to do it. The system should do what people want it to do and
should be safe to operate in different contexts [17]. Don Norman is one of the leading
thinkers in human-centered design and good practice for usability is to follow his design
principles [18, 19]:

- 
  Visibility is the principle that makes an element visible. With more visible elements
  it is more likely that the users know about the element and know how to use it.

- Feedback is the principle that makes it clear to the user that an action has been
taken and a task is executed.

- Constraints is the principle that limiting the interaction possibilities for the user and
guide the user to make the right choice.

- Mapping is the principle that makes the relationship between controls and their
effects they have on the world, clear.

- Consistency is the principle that refers to have similar operations and similar ele-
ments to achieving similar tasks.

- Affordance is the principle that refers that an attribute of an object allows people to
know how to use it. It can also be called as, "giving a clue" of how it should be used.

2.3.2 User experience

Benyon [17] explains user experience as: "Experience design is about recognizing that inter-
active products and services do not just exist in the world, they affect who we are". User
experience design or UX design is basically what the user feels about interacting with a
system, product or service, for example. A website with good UX understand what the user
wants to accomplish, helps the user get there, is simple, easy and make the user feel a sheer
delight when using it. On the other hand, if the user feels frustrated, annoyed or incom-
petent when using a website, the websites UX is bad. And that is why UX is important.
Designers have to ensure that the user gets a positive experience when using the website,
product, system or service [20].
2.3.3 From pen and paper to interactive prototype

By starting with interviews, a better understanding of the requirements of the system can be achieved. From the output, ideas and solutions can be generated and from that, bringing the ideas to life can be done by making for example sketches, storyboard, wireframes etc.

High fidelity prototypes look and feel like the final product. Low fidelity are more focused on functionality, content, form and structure [17].

According to Arnowitz et al. [21] the process of starting with an idea to create a final prototype is defined by the following steps:

- **Plan** - verify the requirements, create a task/screen flow, specifying content and fidelity. Find out and gather the requirements.
- **Specification** - determine the right prototyping characteristics, choose a prototyping characteristics, choose a prototyping method and choose a prototyping tool.
- **Design** - formulate design criteria and create the prototype.
- **Result** - review the prototype, validate the prototype and implement the design.

2.3.4 Designing for developers

Designing and evaluating a useful tool for developers can be a challenge. Measuring how useful a tool is for developers can be a complexed process. For example, if a tool’s purpose is to make it easier for developers to fix bugs, how should that tool be evaluated? Testing the tool efficiency by measure if the bug fixes go faster with the tool than without, might sound like a satisfying solution. But there are more aspects to consider. Does the bug fixes go faster because of the tool, or because of the size of the bug? Or new debugging strategies, or more code knowledge? According to LaToza et. al. [22] when designers are designing a useful tool for developer, they must ensure that the tool solves an important problem and supports the work of the developers by making a strategy faster or more successful. How useful the tool then will be, is depending on both its success in solving the problem, supporting work and how important the underlying problem is to solve. Designing a useful tool require understanding in how the tool supports the developers work and solving an important problem that the developers are facing. This understanding will come with hypothesizes, studies and observations.

2.4 Visual regression tools

2.4.1 Existing tools

There are some existing visual regression tools and in this thesis, three of them are looked into: BackstopJS, Happo and Wraith.

2.4.2 BackstopJS

BackstopJS\(^1\) is taking a screen shot of a page and first sets it to a reference image. After changes, BackstopJS will take a new screenshot called test image and compares it to the reference image. BackstopJS then creates a diff image and visualizes it in a separate HTML page. See figures 2.1 and 2.2.

\(^1\)https://github.com/garris/BackstopJS
Chapter 2. Theoretical framework

Figure 2.1: The interface of BackstopJS, no diffs.

Figure 2.2: The interface of BackstopJS, with diffs.
Pros and Cons

The positive aspects of BackstopJS from this thesis point of view are that it is fast and quite easy to use. The developer just run a command in the command prompt and get the images differences automatically generated. Some negative aspects are that BackstopJS already has an interface which shows the diff images but it is not the way we want to represent it. BackstopJS also uses a configuration JSON file where you specify what URL to check and which components/selectors to take screen shots of. For the developer at Swedbank it would take too much time to test one page at the time.

2.4.3 Happo

Happo helps to find visual bugs before they are merged into the master branch. Happo renders each component, take screen shots of it and finds the differences between that image to a previous image from that component [23]. Happo first runs the current version, usually the latest master branch, and takes screen shots of the components. Then it runs the new version of the code, usually the working branch, where Happo takes new screen shots and compares them with the earlier version. If Happo found a diff, a visual representation will be created and you can then use the diff image to decide if the visual changes are something to fix or not and take the needed actions after that [24].

2.4.4 Wraith

Wraith is made by the developers from BBC News. They had noticed that some small changes happened from release to release and when more team joined projects, small front end bugs tended to come up. Wraith is comparing screen shots of pages at a pull request state when merging to master branch. Wraith is using a headless browser for capturing the screen shots and ImageMagick for comparing the images [7]. ImageMagick is a free software that can create, edit, compose, convert, read and write images in many different ways [25]. Wraith is using PhantomJS as a default headless browser. So, Wraith is taking two images, one usually from a live website and one usually from a local one. The two images are compared against each other, and then a third picture will be created. This image will show the changes in a blue color, even at 1 pixel changes. So, here is where ImageMagick is used then. After receiving the three pictures, you can start to review the changes, which is done manually [7].
Chapter 3

Tools and frameworks

3.1 User Experience Questionnaire

UEQ, user experience questionnaire, is a questionnaire where the participant can express feelings, impressions and attitudes that they feel when using a product [26]. The UEQ enable a fast evaluation of the user experience for an interactive product. The questionnaire contains items that represent two terms with opposite meanings. An example of an item can be seen in figure 3.1. The order of the terms is randomized to ensure that user does not follow a pattern when filling in the questionnaire. Every item is scaled from -3 and +3. 0 counts as neutral, +1 and above counts like a good grade, and -1 and below counts as a bad grade [27].

The UEQ have 26 items and six scales:

1. **Attractiveness**: A general impression, does the user like or dislike the product? Includes the items: annoying/enjoyable, good/bad, unlikeable/pleasing, unpleasant/pleasant, attractive/unattractive, friendly/unfriendly.

2. **Efficiency**: Can the user use the product fast and efficient? Is the interface organized? Includes the items: fast/slow, inefficient/efficient, impractical/practical, organized/cluttered.

3. **Perspicuity**: Is it easy for the user to understand how to use the product? Is the product easy to get familiar with for the user? Includes the items: not understandable/understandable, easy to learn/difficult to learn, complicated/easy, clear/confusing.

4. **Dependability**: Does the user feel he or she has the product under control? Is the product predictable and secure? Includes the items: unpredictable/predictable, obstructive/supportive, secure/not secure, meets expectations/does not meet expectations.

5. **Stimulation**: Does the user feel it is interesting and exciting to use the product?


![Figure 3.1: An item from the UEQ.](image)

**attractive**  o o o o o o o o **unattractive**
Does the user feel motivated to continue using the product? Includes the items: valuable/inferior, boring/exciting, not interesting/interesting, motivating/demotivating.

6. **Novelty:** Is the design of the productive creative and innovative? Is the product getting the users attentions? Includes the items: creative/dull, inventive/conventional, usual/leading edge, conservative/innovative.

A complete UEQ can be seen in Appendix B.1.

The UEQ offers a benchmark that contains UEQ evaluations from 163 products, with a total of 4818 participants. The benchmark classifies the product into 5 categories per scale. These categories are: excellent, good, above average, below average and bad.

### 3.2 Interviews

Interviews are good when briefing and debriefing usability evaluations. There are different interview methods: structured interviews, semi-structured interviews and unstructured interviews. Structured interviews are best when the purpose is to gather data in an organized way and unstructured interviews are used when detailed questions are not planned. A semi-structured interview is a method between structured- and unstructured interview. It allows for some standard questions but also opens up for new questions during the interview [28]. In this thesis, a semi-structured interview method was used.

#### 3.2.1 Semi-structured

Semi-structured interviews open up questions and answers that go deeper than from a more structured interview. The interviewee’s answer might lead the test leader to come up with new questions that the test leader otherwise had not thought of. A semi-structured interview is more demanding by the interviewer, but the effort usually pays off in more deeply understanding of the interviewee’s concerns and perspectives of the tool [29].

In this thesis, a few questions were planned before the interview and was used when needed and for support. Most of the time the tester used followed-up questions. These questions can be seen in Appendix A (the questions are in Swedish).

### 3.3 Likert scale

With a Likert scale, the participants in the test can enter if they agree or disagree with a specific statement [30]. In this master thesis, a 5 point Likert scale was used. In the pre-study phase, a Likert scale was used to find out how developers feel about changing or rewriting code. The alternatives of how often the participant felt insecure was: never, seldom, sometimes, often and almost always. During the final test, the participants answer how much they agree with that the tool could help them feel more secure when they were changing or rewriting code. The alternatives on the Likert scale were: strongly agree, agree, undecided, disagree and strongly disagree [31].
Chapter 4

Method

4.1 Literature Research

In this thesis, a theoretical base was conducted. This thesis includes theory from articles, books and to some degree online. Books and articles were found at Umeå University library\(^1\) and from Google Scholar\(^2\). The theoretical areas that were affected are:

- UX design
- Designing for developer
- Visual regression testing

4.2 Overall Prototyping Process

In this thesis, a prototype of the solution was made. A rapid prototyping process and user-centered design approach were chosen for the prototyping progress. A rapid prototyping process was chosen because the project had a tight schedule and the development process needed to continuing fast.

4.2.1 Rapid Prototyping

Rapid prototyping works well with system development and mainly starts with finding needs and analyze them. Parallel with that is the working process of designing the prototype, evaluate it and implement and maintain the system, going on. Rapid Prototyping works well in cases with new situations when previous experiences are not common \([32]\).

4.2.2 User Centered Design

User-centered design is a design process where the designer involves the end user in the design process \([33]\). By involving the users in the environment that is in question for the product, the result will be more effective, efficient, accepted and successful \([19]\).

\(^1\)http://www.ub.umu.se/
\(^2\)https://scholar.google.com/
4.2.3 The Users

The primary users of the visual regression testing tool are developers working with for example, CSS and web components. The prototype was therefore tested on active developers at Swedbank. According to NNGroup [34], testing on more than five users would not give the tester any more input about the usability. And during the lo-fi evaluations there were only three developers available, therefore the lo-fi prototypes were tested on three users (Java specialists). The final prototype was tested on five developers, working with UI development.

4.3 Pre-study

To determine the needs, problems and requirements, a pre-study was made. The pre-study phase included a workshop with two UI developers and a questionnaire that was sent out to developers at Swedbank.

4.3.1 Workshop

During the workshop, the problems were stated and some possible solutions were proposed. During this meeting, we realized that finding a complete solution for all the requirements (automation, showing relevant images or be able to approve or fix diff images for example) is very complex. CSS or other code changes can be executed from different directions and all the changes that can generate diff images might affect each other. The requirements were therefore scaled down to a more suitable solution for this thesis.

4.3.2 Questionnaire

To state if the prototype can help the developer feel more confident to change code (and decrease fear-driven development) a short questionnaire was made. The 5-points Likert scale was used and the survey was sent to the developers at Swedbank by email and 26 developers answered the questionnaire. Since not all of the developers at Swedbank are Swedish-speaking, the questionnaire was written in English. The questionnaire can be seen in Appendix C.1.

4.4 Low fidelity prototype

After the pre-study workshop and when the problems were stated, a low fidelity (lo-fi) prototype was made and tested. A lo-fi prototype is useful when you want to test functionality and do not want the user to focus on aesthetic values. A first lo-fi was made and shown for two developers from the UI team. The prototype was discussed and a new improved lo-fi prototype was created and tested on three developers at Swedbank to see if the desired functionality and design were reached. During the lo-fi testing, a think-aloud method was used, which means that the user speaks out loud what he or she feels and thinks when using the system [30]. The test subjects were given a case to solve when using the prototype. The prototype was made interactive, i.e. if the user “clicked” on a drop-down menu, another small piece of paper was placed at that spot, see figure 4.1.
4.5 High fidelity prototype

After the lo-fi prototype had been tested and evaluated, a high fidelity (hi-fi) prototype was made. The hi-fi prototype was not tested with user tests. This choice was made because a rapid prototyping process was used, i.e. the process is continuing fast, but also because the most important part of the prototype is that it has its required functions to be a useful tool for the developers. Therefore a hi-fi user test was not highly prioritized. The hi-fi prototype was discussed with the purchaser of the tool though, and improvements were made. The hi-fi was also made to easier implement the final prototype.

4.6 Final prototype

4.6.1 Implementation

The final prototype was implemented with a front end part in AngularJS and a back end part in NodeJS. Mock data were used for the screen shots taking part. Therefore it is still called a prototype and not a fully developed system. The image comparison was implemented though and made with ResambleJS. More of these techniques in section 4.7.

4.6.2 Proof of concept

To make sure ResambleJS could work for our purpose, a small test was done to find out how small changes it could handle. By starting with an ordinary image of a mug, a smiley was drawn on the mug in Paint (a drawing program) and both images were compared with ResambleJS. The script marked the difference and the mismatch percentage was 0.02, see figure 4.2.
4.6.3 Pilot test
Before the main study, a pilot test was conducted to test that the experimental method was working and that the application did not have any bugs [30]. The pilot test was made on another master thesis student at Swedbank and after the test, some small changes were made. The changes included layout changes as font family and font size changes and one bug fix.

4.6.4 Study design
Five developers were given a case where they had changed the size of a label and wanted to see how the change was affecting the rest of the site by using the prototype. The participants were asked to think-aloud during the test. Afterward, a small semi-interview was held to figure out how the interface felt for the participant and if the tool could be helpful for the developers. They also used a 5 Likert scale to show how much they agreed if the new tool could be useful for them in their developing process and decreasing the fear of code development.

4.6.5 Evaluation
The final prototype was evaluated by its functionality and usability. By using the UEQ the prototype’s usability was evaluated. A semi-structured interview was held with the participant to obtain valuable feedback from both the design and functionality of the tool. The test subjects were also asked to state, with a Likert scale, how much they felt that the tool could be helpful for them during their developing process and when they are changing code.

4.7 Tools
For the lo-fi prototype, pen and paper were used and the hi-fi prototype was made in Axure-eRP 8\(^3\) (a desktop application for creating interactive protypes). The final prototype was

\(^3\text{https://www.axure.com/}\)
4.7. Tools

implemented in WebStorm\(^4\) (an integrated development environment for JavaScript development) by using AngularJS\(^5\) (a JavaScript framework) in the front end part while the back end, that integrated the front end part with image comparisons, was done with NodeJS\(^6\) (a server framework). ResembleJS\(^7\) (a JavaScript library for comparing and analyzing images) was used for the image comparing process.

\(^4\)https://www.jetbrains.com/webstorm/
\(^5\)https://angularjs.org/
\(^6\)https://nodejs.org/en/
\(^7\)https://huddle.github.io/Resemble.js/
Chapter 5

Results

5.1 Pre-study

The result from the pre-study phase, which included a workshop and a questionnaire, is reported in this section.

5.1.1 Workshop

During two meetings with UI developers at Swedbank, the requirements for the final prototype were stated:

- A test interface.
- Integrate with selenium tests.
- Compare two images and generate a diff image.
- Save settings.
- Save a test.
- Ignore diff images.
- Get diff image and URL link.
- Choose portal/portal-branch/app/app-branch.
- Filter by apps/error-percentages/Selenium tests/devices/browsers.
- A view that displays the same version but different devices.
- A view that displays the same device but different versions.

The bolded requirements are the ones that were fulfilled in this thesis.
5.1.2 Questionnaire

A questionnaire was sent out to find out how developers at Swedbank feel about changing code. 26 participants answered the questionnaire, and of these participants, more than 50% had been working at Swedbank one year or less. When changing in the code, the average feeling of how developers feel, was that they sometimes felt fear of changing code. The fear was bigger when they were changing a global web component, see figure 5.1. However, the developer still felt quite sure that the quality of the website is retained even though they do not feel certain that they have looked through every page the change might have affected, see figure 5.2.

5.2 Low fidelity prototype

The lo-fi prototype can be seen in figure 5.3. After lo-fi testing, some inputs from the participants were received. On the run test view, two of the test subjects felt that saving settings were unnecessary, they would only have saved the whole test. To be able to choose multiple or all devices was a desirable feature. On the diff image view, all participants thought the visualization of the resulted diff images could be clearer. They wanted more information about the diffs, such as where the images are coming from and what have been executed. One diff image row should represent a view and if the user wants to ignore a diff image, there should be a button on every row.

5.3 High fidelity prototype

The first iteration of the hi-fi prototype can be seen in figure 5.4 (the run test view, where the developer configures a new test) and in figure 5.5 (the diff images view, where the developer can see the diff images that occurred). After discussions with the purchaser of the tool, modifications on the hi-fi prototype were done. Choosing device and browser part was redesigned to better fit in with technical criterias and placed last in the run test view. The feature to save settings were removed when the purchaser did not see it as necessary anymore. The diff images got more space on the view to get bigger and more visible. The information text (with URL, app, etc.) were therefore placed under the pictures instead of beside, and placed in an expandable box. The second iteration of the hi-fi prototype can be seen in figure 5.6 (the run test view) and in figure 5.7 (the diff images view).

5.4 Final prototype

The final prototype can be seen in figure 5.8 (the run test view) and 5.9 (the diff image view).

5.4.1 System description

The final prototype was implemented with AngularJS in the front end part and with NodeJS in the back end part. On the run test view, the developer can choose to either use a saved test or customize a new test. When customizing a new test, the developer first chooses current settings, i.e. which portal, portal branch, app and app branch the developer currently are working on. When the developer has chosen one app, another row appears where the developer can choose another app if he or she wants to test more than one app. After chosen
In general, do you feel certain that either nothing brakes or some visual errors occur when changing the code in a web component?

![Bar chart]

If you changing the code in a global web component, do you feel certain that the change won't give any visual error somewhere you're not aware of?

![Bar chart]

Figure 5.1: Survey picture 1.
Do you feel certain that the quality is retained on the site you changed the component at?

Do you feel sure you've controlled all the pages the component is used on after the change?

Figure 5.2: Survey picture 2.
5.4. Final prototype

5.4.2 Usability tests

From the usability test and semi-structured interviews, the participants in overall had a good experience with the tool. All of the participants experienced fear-driven development and they all thought this tool could be helpful for them to feel more secure when changing or rewriting code. They gave their agreement of how it could help them between 3 and 5 on the Likert scale, with a mean of 3.8. Three of the participants would have given higher rate
Figure 5.4: High fidelity prototype, the run test view, first iteration.
Figure 5.5: High fidelity prototype, the diff images view, first iteration
Figure 5.6: High fidelity prototype, the run test view, second iteration
Figure 5.7: High fidelity prototype, the diff images view, second iteration
Figure 5.8: Final prototype, the run test view.
Figure 5.9: Final prototype, the diff images view.
if they knew the Selenium tests was good enough. If they used this tool today, they would not feel entirely certain when doing code changes. That is not all because of the tool though, because they know that the Selenium tests are not developed enough. With better Selenium tests they would feel more secure and give a higher rate. Three participants thought the filter functions in the diff image view could be improved. The filter was programmed to show pictures with any of the filter criteria, but the participants wished that it should only show images with all of the filter criteria that were chosen.

All participants thought the interface had a nice and clean layout but they faced some troubles in the run test view. Some misunderstood the function to add a new app with the app they want to compare to. And all of them thought the layout was nice, but the steps (1, 2, 3) took a little bit too much space. The steps were good the first time, but because the developers probably are going to use the application often, the steps are not necessary.

Then the participants had some own wishes they came up with, features that they though would have been nice to have.

– See the chosen branch after URL text.
– Click for bigger images in the diff view.
– Set app branch default as portal branch.
– One more overall diff view with thumbnails of all diff images.
– Somehow highlight changes that are more unconscious.
– Combine with metadata.
– Set at default for device/browser.
– Save test also from diff image view, in case you forgot to save it from run test view.
– Add device and browser in the information part of the diff row.
– Choose apps with check boxes instead.
– Better name for ”Views run”, for example ”screen shots taken” could be better.
– More impersonal text, instead of ”Your result” it should be, ”The result”.
– Better diff images that can show what the error is. Is it a spelling mistake? Another size? Circle the error instead of color it would have made that possible.
– Use animations for making the application feel more modern.
– be able to click on URL-heading and there be able to see how the diff looks over all browsers.
– Make it easier to choose ”the latest” of browsers, i.e. the latest version of Chrome, Safari, Firefox in latest Windows and Mac.
5.4. Final prototype

5.4.3 UX questionnaire

The result from the UX questionnaire shows that the users had a good experience of the application. Figure 5.10 shows the positive or negative experience of the application. The blue bar is the mean value and the black bar shows the confidence interval of 5%, which means that the true value will be in this range. To be seen in the figure, the application obtains positive experience from all of the participants. From the benchmark, see figure 5.11, it shows that compared with other systems and products, this application has a high range of attractiveness, perspicuity, efficiency, dependability, stimulation, and novelty.
Chapter 6

Discussion

From the pre-study questionnaire, it is quite clear that many developers suffer from fear-driven development. They do not feel certain when changing code because they do not know how the change will affect the end result of the website or application they are working at. One code change can lead to hundreds of visual errors. In this master thesis, a tool for showing visual changes for the developers is created. The tool got positive feedback from the participants that tested it and all of the participants felt that this tool could help them feel more secure when making code changes and decrease their feeling of fear-driven development.

From usability guidelines, an application should be easy to follow, learn and the user should know how to use it, to achieve a good usability. These guidelines were taken into account when designing this tool and steps, or instructions, that the developer could follow were therefore designed. But for the developers that tested the application, they all thought the instructions were good the first time, but when they already know the tool, the steps were just in their way. So from a new point of view, it could have been better and more relevant for a developer, if they could choose to see instructions or not. For example, implement a question mark that expands the information when hovering over it. That would also have taken less space.

In the beginning of the testing part, only the lo-fi prototypes were tested. The hi-fis were not tested because the most important part of the tool was classified to the functionality, and not the layout of the application. And with a rapid prototyping process, we wanted to move on fast and begin with implementation and set up the development environment. Therefore, the hi-fi prototypes were not tested on developers but discussed with the purchaser who came with input. During the final prototype tests, there came a lot of inputs from the participants, and maybe some of these inputs could have been avoided if the hi-fi prototype was tested as well. But of course, it is not certain. We experienced big differences of input from the lo-fi prototype testings and final prototype testings, but it could also been because of the participants did not felt it was finished during the earlier state of the prototype testings. An interesting subject to talk about is how usability testing varies from "normal" users, and developers. Does a user, that also is a developer, see an application differently, compared with a user that does not work with programming? Perhaps the inputs from the final prototype have not occurred during hi-fi prototype testing because it is still not a "finished" tool.

This tool compares images and shows a diff image for the developers if there are any changes in the visual interface. But one problem that is not completely solved in this
application, is how to avoid getting diff images when data is changed. It could happen that it is not the CSS or a web component that are changed and that a diff image occurs when, for example, data from a database has been modified. This is a diff image the developer does not care about. It is not a visual error but will act like one.

6.1 Limitations

This thesis was limited by the amount of time. All the features and requirements the purchaser and other developers had, have not been implemented because of the time limit. During the developing process, the project had troubles with an outdated NodeJS version. We suffered some technical issues because of that and it took a lot of time as well. The distance to the purchaser was also a limitation when most of the communication was held over email which increases the risk of misunderstandings.

6.2 Future Work

The tool has a lot of features that can be continuing working on and implemented. There are also improvements that can be done with the existing tool. Is there any better way to show the diff images? How would it work in real-life situations when the developer have 100 images to working through? And what about the human eye? It is still possible that the developer misses out a changing when it is 100 images to look through? These questions are just a few to ask when a future developing process of the tool continues. Other work that also can be considered in the future is to discuss if the Selenium tests are the best choice for taking the screen shots.
Chapter 7

Conclusions

This thesis is investigating the current fear-driven development that exists among developers and studying if a visual regression testing tool can be helpful for the developers during the evolving process of the application.

Changing CSS or other code can result in big visual errors. It is common that same web components and CSS are used on several places in a website or application, which make the developers more uncertain how the changes he or she does, will affect the whole site. From the pre-study phase in this thesis, the study shows that fear-driven development is a usual feeling for developers when they feel uncertain when changing code sometimes or more often. In this thesis, a visual regression testing tool was made to study if it could be a helpful tool for the developers. The tool can take screen shots of the site, before and after the developer making the change, compare the screen shots with each other and show the differences that might have occurred for the developer. The tool was tested on UI developers to see if it could support them and make them feel more certain when changing code, and the results were positive. The developers thought it was a great tool and all of them estimated the tool as helpful for them and it made them feel safer when making code changes. There are still some improvements to be done on the tool though, that can enhance the developers’ fear-driven development even more. With more functionality to the tool that suits the developers needs and with better Selenium tests that are integrated with the tool, the developers would give the tool a higher grade than in the current state, but the response about the tool was still positive and the developers still felt more secure when changing code using this tool, than not using it.

When the tool had mock data instead of real taken screen shots, this study can not certainly prove the helpfulness it can offer to the developers. And also, in real time situations, the diff images that generates can be much more than the ones that were shown to the developers in the tests. Still, the tool has potential, and the feedback from the developers during the tests can improve the tool to work even better.
Chapter 8

Acknowledgements

I want to say a great thank you to my supervisor Markus Johansson for guidance, support and expertise sharing when working on this project. Also, thanks to Mårten Norén who gave me the requirements of the application and support during the developing process. And thanks to Swedbank that offered and provided this work.

I also want to thank my supervisor Kalle Prorok, at Umeå University for guidance in writing this thesis.

Finally, I want to thank my peer-reviewees Amanda Dahlin, Lena Lundgren, Mikael Whalström and Robin Schulze, for coming up with valuable feedback on the report.
References


Appendix A

Semi-structured interview questions

The questions used as a basis for the semi-structured interviews is listed below. When the interviews were held in Swedish, the questions are also in Swedish.

1. Upplever du rädsla för att ändra kod?
   (a) Varför? Hur? När?

2. Upplever du svårigheter att hitta visuella förändringar när du ändrar kod?
   (a) Hur gör du nu?

3. Hur tror du att du skulle kunna bli hjälp med att känna mindre rädsla samt hitta förändringar?

4. Hade detta verktyg hjälpt?
   (a) Hur hade detta verktyg hjälp dig som?
   (b) Eller stjälpt?

5. Hur upplevde du att det var att använda verktyget?
   (a) Lätt? Svårt? Tråkigt? Kul?
   (b) Varför? Varför inte? Vad saknas? Vad var bra?

6. Hur upplevde du gränssnittet?
   (a) Fint? Fult? Kreativt? Roligt?
   (b) Varför? Varför inte? Vad saknas? Vad var bra?
Appendix B

UEQ - user experience questionnaire

The user experience questionnaire, UEQ in figure B.1.
Please assess the product now by ticking one circle per line.

<table>
<thead>
<tr>
<th>annoying</th>
<th>enjoyable</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>not understandable</td>
<td>understandable</td>
<td>2</td>
</tr>
<tr>
<td>creative</td>
<td>dull</td>
<td>3</td>
</tr>
<tr>
<td>easy to learn</td>
<td>difficult to learn</td>
<td>4</td>
</tr>
<tr>
<td>valuable</td>
<td>inferior</td>
<td>5</td>
</tr>
<tr>
<td>boring</td>
<td>exciting</td>
<td>6</td>
</tr>
<tr>
<td>not interesting</td>
<td>interesting</td>
<td>7</td>
</tr>
<tr>
<td>unpredictable</td>
<td>predictable</td>
<td>8</td>
</tr>
<tr>
<td>fast</td>
<td>slow</td>
<td>9</td>
</tr>
<tr>
<td>inventive</td>
<td>conventional</td>
<td>10</td>
</tr>
<tr>
<td>obstructive</td>
<td>supportive</td>
<td>11</td>
</tr>
<tr>
<td>good</td>
<td>bad</td>
<td>12</td>
</tr>
<tr>
<td>complicated</td>
<td>easy</td>
<td>13</td>
</tr>
<tr>
<td>unlikable</td>
<td>pleasing</td>
<td>14</td>
</tr>
<tr>
<td>usual</td>
<td>leading edge</td>
<td>15</td>
</tr>
<tr>
<td>unpleasant</td>
<td>pleasant</td>
<td>16</td>
</tr>
<tr>
<td>secure</td>
<td>not secure</td>
<td>17</td>
</tr>
<tr>
<td>motivating</td>
<td>demotivating</td>
<td>18</td>
</tr>
<tr>
<td>meets expectations</td>
<td>does not meet expectations</td>
<td>19</td>
</tr>
<tr>
<td>inefficient</td>
<td>efficient</td>
<td>20</td>
</tr>
<tr>
<td>clear</td>
<td>confusing</td>
<td>21</td>
</tr>
<tr>
<td>impractical</td>
<td>practical</td>
<td>22</td>
</tr>
<tr>
<td>organized</td>
<td>cluttered</td>
<td>23</td>
</tr>
<tr>
<td>attractive</td>
<td>unattractive</td>
<td>24</td>
</tr>
<tr>
<td>friendly</td>
<td>unfriendly</td>
<td>25</td>
</tr>
<tr>
<td>conservative</td>
<td>innovative</td>
<td>26</td>
</tr>
</tbody>
</table>

Figure B.1: UEQ.
Appendix C

Pre-study questionnaire

The pre-study questionnaire about fear-driven development sent out to developers, see figure C.1.
Changing web components

Are you a developer at Swedbank? Do you work with web components? Then you're the right person to answer these questions, it takes less than a minute!

1. For how long have you been working at Swedbank?
   Mark only one oval.
   
   < 1 year
   1 year
   2 years
   3 - 5 years
   6 - 10 years
   > 10 years

Quality assurance when changing web components

When changing a web component, the change can affect many pages and visual errors might occur. These questions are made to find any patterns in how a developer thinks about changing web components and how it will affect the website.

2. In general, do you feel certain that either nothing brakes or some visual errors occur when changing the code in a web component?
   Mark only one oval per row.
   
   Never  Seldom  Sometimes  Often  Almost always
   

3. If you changing the code in a global web component, do you feel certain that the change won't give any visual error somewhere you're not aware of?
   Mark only one oval per row.
   
   Never  Seldom  Sometimes  Often  Almost always
   

4. Do you feel certain that the quality is retained on the site you changed the component at?
   Mark only one oval per row.
   
   Never  Seldom  Sometimes  Often  Almost always
   

5. Do you feel sure you've controlled all the pages the component is used on after the change?
   Mark only one oval per row.
   
   Never  Seldom  Sometimes  Often  Almost always

Figure C.1: Questionnaire to the pre-study phase.