Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Process Analysis Nr. 6

Automated GUI Style Guide Testing

Marcus Eisele, Christan Endres, Matthias Fetzer

Course of Study: Softwaretechnik

Examiner: Prof. Dr. Stefan Wagner

Supervisor: Dipl.-Inf. Ivan Bogicevic

Commenced: April 15, 2014

Completed: October 15, 2014

CR-Classification: D.2.5

Abstract

This process analysis evaluates the existing GUI development and testing process of a multinational IT company and tries to find rooms for improvements. This company develops a program which provides a web front end for users. At the moment there are no programmatically usable style guide and no automated test cases for the GUI used.

Thus to enhance the current processes we describe a set of characteristics and principles to aid in the creation of a pattern library. Additionally we evaluate several tools to support automated testing of the created components and the implementation of the product in question. Finally we show how the results can be integrated into the existing development process.

Contents

List of Figures

List of Tables

1 Introduction

The following sections describe the goals and the underlying conditions of this process analysis.

1.1 Module Process Analysis

The goal of a process analysis is to study and evaluate a company's processes (or exerpts of its processes) and find rooms for improvement. The students analyze the current condition and present their suggestions. The suggested improvements do not necessarily assume that the suggestions describe the ideal solution but rather illustrate the most feasible solution.

1.2 Industry Partner

The industry partner involved in this process analysis is a multinational technology and consulting company. The headquarters of the company are located in the United States. Due to the multinational organization and structure as well as the size of the company, the industry partner can be described with the following criteria:

- **Industry**: The company operates in the fields of computer hardware, computer software, IT services and IT consulting. The department which we worked with has its main focus on computer software and software development.
- **General Structure**: The company has numerous locations across the globe and therefore has a strict/clear seperation among its business fields.
- **Organization**: Due to the size of the company, its organization contains very deep hierarchies and long official channels. This results in very limited and dense interaction between the departments.

We collaborated with a development team whose principal activity is to develop a web based business application. Therefore the focus of this process analysis lies on the development teams tools and processes.

1.3 Desired Improvements

The mentioned industry partner already uses tools to test the functionality of a specific web application against the requirement specification. However the visual appearance defined in the style guide is not covered by these tests. The goal of this process analysis is to evaluate how the visual appearance can be automatically tested. Furthermore it is of interest how the visual testing steps can be embedded into the existing processes.

1.4 Project Planning And Schedule

The following sections describe the planning and schedule of the process analysis.

1.4.1 Work Packages And Gantt Diagram

This section gives insight into the planning of the process analysis. The Table 1.1 describes the work packages and tasks and the Figure 1.1 visualizes the distribution over the time.

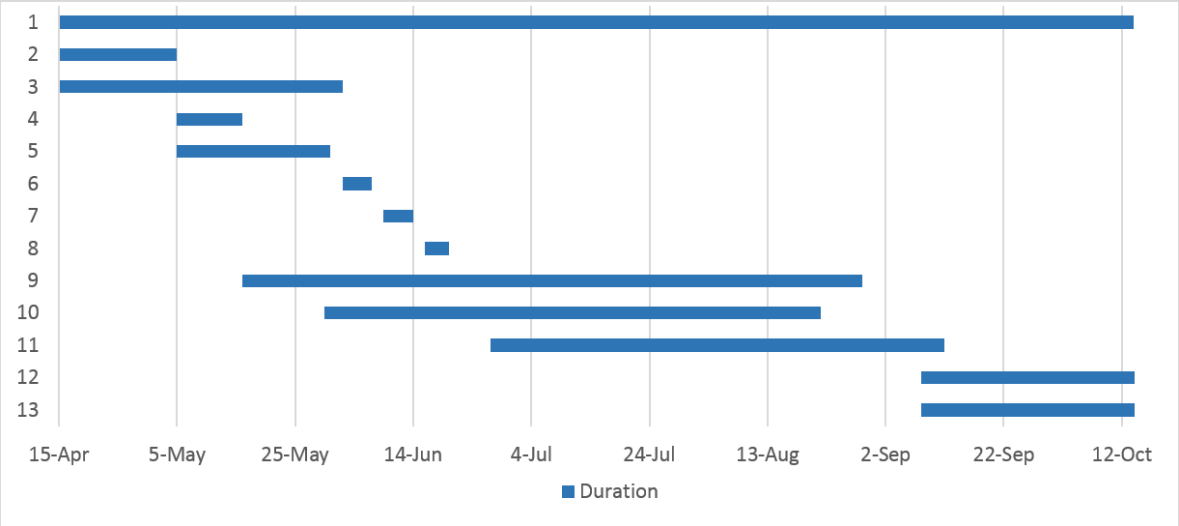

Figure 1.1: Gantt-Diagram

1.4.2 Responsibilities

Table 1.2 describes which team member mainly worked on which taks.

	Package	Tasks	Start	End	Term
1	Project	Whole project	2014/04/15	2014/10/14	182
2	Data Collection	Planning the project, Kickoff, Interviews	2014/04/15	2014/05/05	20
3	Framework Research	Research of possible Frameworks	2014/04/15	2014/06/02	48
4	Process Analysis	Collect data	2014/05/05	2014/05/16	11
5	Data Analysis	Analyse data	2014/05/05	2014/05/31	26
6	Presentation Preparation	Prepare the presentation	2014/06/02	2014/07/06	5
7	Pentecost		2014/06/09	2014/06/14	5
8	Presentation	Give the presentation	2014/06/16	2014/06/20	4
9	Documentation	Write document	2014/05/16	2014/08/29	105
10	Framework Test	Test frameworks and define criteria	2014/05/30	2014/08/22	84
11	Framework Comparison	Prepare release candidate	2014/06/27	2014/09/12	77
12	Finalizing	Finalize the print version of the document	2014/09/08	2014/10/15	37
13	Presentation	Prepare and give the final presentation	2014/09/08	2014/10/15	37
	Time buffer planned not to use		2014/10/01	2014/10/15	14

Table 1.1: Work packages of the project

Tasks	Christian	Marcus	Matthias
Planning the project	•	•	•
Define first questions	•	•	•
Research Test Frameworks		•	
Research pattern library Frameworks	•		•
Collect data due questioning			•
Analyse data	•		•
Prepare first presentation	•	•	•
Give the first presentation	•	•	•
Prepare LaTeX templates	•		
Prepare graphics	•		
Prepare collaboration platform and tooling			•
Define framework criteria	•	•	•
Test frameworks		•	
Write document chapters accordingly	•	•	•
Prepare final presentation	•	•	•
Give final presentation	•	•	•

Table 1.2: Responsibilities of each group member

2 Current Condition

In addition to the characteristics described in section 1.2 the following sections describe further parameters and constraints of the examined development process. The current condition has been ascertained through interviews with the industry partner.

2.1 Teams

There are four teams involved in the web application development process. Each team is working at a different location. In this section we point out their responsibilities in the context of this process analysis.

2.1.1 User Experience Team

The User Experience Team (hereinafter referred to as the UX-Team) consists mainly of one graphic designer who creates the style guide according to the requirement specification. The designer is responsible for testing the web application against the style guide. Therefore the UX-Team reports violations against the style guide to the Developer Team.

2.1.2 Developer Team

The Developer Team develops the web application and corrects defects in the web application. They receive the requirement specification and style guide and use them as the nominal condition. They also create user stories which are used by the Testing Team to test the functionality of the web application.

There are no dedicated developers for the graphical user interface. The whole team is involved in all aspects of the development process.

2.1.3 Testing Team

The Testing Team tests the functionality of the web application according to the previously defined user stories. The tests verify the functionality according to the requirement specification. Occurring conflicts or defects get reported to the Developer Team.

2.1.4 Product Management

The Product Management mainly develops the requirement specification document. During the development phase they report defects to either the UX-Team or the Developer Team.

2.2 Artifacts

This section describes the artifacts which are produced or used during the development process.

2.2.1 Requirement Specification

The requirement specification describes the nominal condition of the web application. The document itself is of little interest for this process analysis because the focus is on the automation of the GUI testing process.

2.2.2 Style Guide

The style guide consists of multiple documents, each describing one or multiple parts of the web application appearance. The style guide is provided as a PDF document, containing textual and graphical descriptions of the web application. The style guide is provided via the intranet. Each style guide document has five section types described in table 2.1.

2.2.3 User Stories

The user stories serve the Testing Team to test the functionality of the web application. Because the user stories are not involved in the process of testing the graphical user interface, they are of little interest for this process analysis.

Type	Description
Identification	This section identifies the part of web application which is referenced by the document. It contains information about the product name, application part name and the date of the last update. This section appears exactly once in each document.
Changelog	The changelog provides information about each change inside the document. Each change is described by a change description, the page(s) on which the change appears and the date of the change. This section appears exactly once in each document.
Colors and fonts	This section describes global values for colors and fonts in the document. The colors have a name, are shown as a square and are described by its hex and rgb values. The fonts are named and described by its appearance, size and color. Also the sizes are defined by point-, EM-, pixel- and percentage-values. This section appears exactly once in each document.
Overview	This section shows an overview of a component described in detail by the following section type “Detail View”. There is at least one big picture of the component and maybe multiple pictures in different screen resolutions. One example would be the “Title Page”. This section appears at least once in each document.
Detail view	This section describes one component of the overview section in detail. Part of the detailed descriptions are mostly multiple small pictures of the component and hints for the implementation like sizes, gaps, colors and fonts. One example would be the “Footer” of the “Title Page”. This section appears at least once for an overview section in each document.

Table 2.1: Section types of the style guide

2.2.4 Defects

Defects are violations inside or against the requirement specification or style guide. Defects are assigned to either the UX-Team or the Developer Team depending on the specific issue.

2.2.5 Web Application

The developed web application is a cloud-based integrated software suite for human ressource management. Technically it consists of a web based java application with a java backend. The features and requirements are specified in the requirement specification. The appearance

is defined in the style guide. The Developer Team uses these documents to create this application.

2.3 Used Applications

The style guide is created with Adobe InDesign and Adobe Photoshop. The Developer Team works with Eclipse IDE, Rational Team Concert and IBM Mantis.

2.4 Development Processes

This section describes the development of the web application, the interaction between the teams and their responsibilities.

2.4.1 Modified Scrum Process

The Developer Team is developing with an adapted Scrum process. Figure 2.1 shows an overview of the development process itself. The task responsibilities of the teams are shown on the y-axis. The x-axis represents a sprint duration of six weeks.

Note that the the planning of the next sprint is before the start of the next cycle. The testing phase of the Testing Team starts two weeks after the start of the development phase. The creation of the style guide and the testing against it by the UX-Team does not comply with the scrum phases.

Integration Day

One feature of the modified scrum process is the integration day. During the four weeks of development there is one integration day each week. The current code is deployed on a test environment where the whole Developer Team test their code by hand.

Promotion Day

At the end of the last development week there is a promotion day. All code without known issues gets promoted to the next higher code stream.

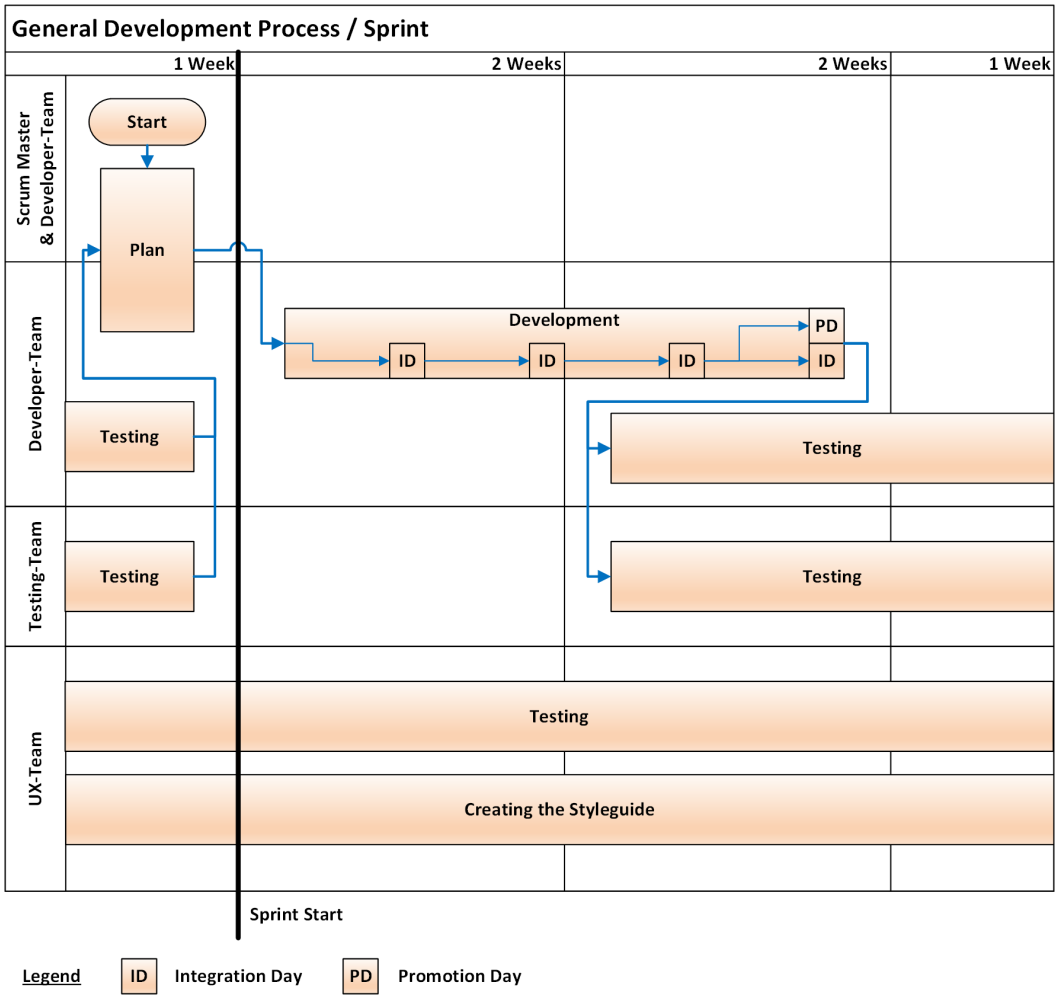

Figure 2.1: Scrum Sprint Cycle

Code Streams

There are three code streams to manage the maturity of developed code. Each of the lower code streams can be promoted at the promotion day. There are three code streams:

- *Development Stream (SW10)*: This code stream is the first and lowest in which the code of the current sprint is stored.
- *Staging Stream (SW20)*: The second code stream contains the code which runs on the staging servers.
- *Production Stream (SW30)*: The third and highest code stream which contains the code running on the production servers.

2.4.2 Roles And Responsibilities

There are five roles involved in the scrum process. They can be described as follows:

Product Management

The Product Management is responsible for creating the requirement specification.

Scrum Master

The Scrum Master takes care of the sprint planning. He creates the user stories and work items in Rational Team Concert based on the requirement specification. Further he estimates the effort and priorizes the work items together with the Developer Team. Prior to each sprint he assigns work items to developers.

Developer

The main task of each developer is to complete their work items. This is done by the following procedure:

1. Develop code using Eclipse IDE and Rational Team Concert.
2. Compile and pack the code using IBM Mantis.
3. Deploy the code to a local virtual machine for smoke testing.
4. Do smoke tests.
5. Review the code.
6. If there are no problems deliver the code to the stream.

The testing and reviewing by the Developer Team is not the testing shown in figure 2.1, but rather part of the web application development.

Tester

The tester is responsible for testing the functionality of the web application. To ensure that the current user stories have been implemented successfully, the tester launches Accessability Verification Tests, Build Verification Tests and Globalization Verification Tests. They do not test whether the web application is in compliance with the style guide or not.

UX-Team

The responsibility of the UX-Team covers the creation and maintenance of the style guide. Furthermore the UX-Team verifies the compliance of the web application's visual appearance with the style guide.

3 Analysis For Potential Improvements

This section describes the issues which have potential to be improved.

3.1 Issues With The Style Guide

The following section unveils the issues with the style guide, regarding the communication, the format and its technical implementation.

3.1.1 Communication

The UX-Team creates or rather regularly updates the style guide. The style guide is published on the intranet. There are no update notifications for the Developer Team.

Another issue is the conflicting terminology. The UX-Team is mainly used to graphic designer terms, while the Developer Team is used to web design and development terms. Thus the language of the UX-Team and the style guide is influenced by Photoshop and has to be translated to the web development terminology.

3.1.2 Style Guide Format

The style guide consists of multiple pdf files. There is no version control to handle the changes. Thus there is no comfortable way of showing the differences between several versions. Furthermore the overall size of the style guide is huge compared to the functionality it provides.

Another issue is the gap between the static representation with screenshots of the style guide and the dynamic behavior of the graphical frontend of the web application. The style guide does not always cover each state of appearance of the frontend with screenshots and sometimes tries to describe it with text.

3.1.3 Change History

The change history of the style guide is often ambiguous. Changes are not necessarily reflected in all documents or even not in all sections of a style guide document. Therefore inconsistencies are going to occur across documents.

3.1.4 Duplicates

Due to the format of the style guide and the information provided in each pdf document there are several duplicated information in many different places. These information may or may not be consistent. There is no utilization of reusable components. This not only makes the maintenance of the style guide harder, but it also complicates the development as well.

3.2 Issues With Testing Against The Style Guide

This section enumerates the current issues with the testing against the style guide.

3.2.1 Testing By The Developer Team

The Developer Team manually tests the visual appearance of the web application. This is very time consuming and error-prone.

Another issue is the non-formal format of the style guide which prevents automated testing against the style guide.

3.2.2 Testing By The UX-Team

The UX-Team also tests the visual appearance web application against the style guide. These tests do not happen on a regular basis and the feedback gets reported at any time.

4 Nominal Condition

This chapter describes the nominal condition regarding the documents, the processes and the degree of automation.

4.1 Documents

According to the findings in section 3.1 an improved format for the style guide is needed. Ideally the new style guide format has to fulfil the following criteria:

- **Technical Format**: The new format should be more technically oriented, in a way that the style guide can be easier maintained and updated. Ideally the new format allows version control and automatically creation of a changelog, as well as a better documentation in general.
- **Unambiguousness**: The new format should be very clear and unambiguous. Therefore the new version of the style guide should include concrete code samples on how to use the components. This allows the developers to reuse code snippets and therefore avoids multiple interpretations of the style guide.
- **Avoidance of duplicates**: The new format should try to avoid duplicates of any kind inside the style guide. Thus there is a need of reusable components which can be referenced inside the style guide. This ensures that updates to the style guide are distributed to all sections. Besides the avoidance of duplicates (and therefore inconsistencies) inside the style guide itself, it will also help to avoid duplicates and inconsistencies in the resulting product.

4.2 Degree Of Test Automation

In the current condition there is no automated testing of GUI components. If a developer wants to ensure that his changes have no side effects on existing pages or components, he has to check every page before committing his changes. This is not impossible, but has a very low benefit-cost ratio and is therefore just not viable for every change. A developer spends a lot of time to test the impact of even small changes because the impact could occur almost anywhere.

Therefore it would be beneficial to test regressions of the GUI or other side effects affecting the GUI automatically.

A general conflict is that the tests done by the Developer Team have not to be time consuming but large tests are usually long running. Thus a Developer will lose a lot of time or may just skip the tests altogether. But to use the whole potential of screenshot based testing a lot of tests may be necessary.

In addition to the smoke tests of a developer which are done relatively fast, there is a need of a full test which automatically detects impact of changes over the whole project.

4.3 Communication And Processes

The communication between the Developer Team and the UX-Team happens mainly through the exchange of the documents. Thus the requirements described in section 4.1 have to be implementable into the processes of both departments. The same counts for a proposal of introducing an automated test as described in section 4.2.

The main focus has to be on finding a solution which can be introduced almost exclusively by the Developer Team because the possibility of proposing changes to the UX-Team seems to be very limited.

5 Pattern Libraries

To address the criteria mentioned in section 4.1 a new style guide has to be created. The best solution would be the UX-Team delivering such a style guide. As this most likely will not be the case, an intermediate form of a style guide has to be created. The intermediate format can be some sort of component library. To distinguish between those documents we introduce the term pattern library as it has been established in research and industry. [MLW05]

5.1 Characteristics Of A Pattern Library

A good pattern library can be characterized by the following criteria:

- **Searchable / browsable**: A good pattern library includes a proper navigation to ensure that developers and designers easily find what they are searching for.
- **Component based**: A good pattern library supports the creation of reusable components. These components can be used across several projects and within the pattern library itself. In the best case a pattern library supports cross referencing between components as well as creating component groups.
- **Samples**: A good pattern library directly shows samples of the components in question. The developer and the designers directly see how the described component looks like in production. In the best case a pattern library supplies several presentations of the component in question. For example the pattern library presents how the component looks in desktop and mobile web clients.
- **Code samples**: A good pattern library allows the developer to directly copy code snippets of the selected component including markup code such as HTML and CSS as well as the backend code in question. This ensures that the component will exactly look like it has been defined in the pattern library. In the best case several code samples are provided, e.g. in different programming languages or for different clients.
- **Supplemental commentary**: A good pattern library includes a reasonable amount of supplemental commentary. Examples for this kind of commentary are: When to use the component, when not to use the component, similar components or related components.

5.2 Pattern Library Examples

Many Pattern Libraries are created without or with not published tools. Thus we cannot provide a list of good tools which support the creation of a pattern library. Therefore we supply a list of good implementations of Pattern Libraries as a reference. All of these Pattern Libraries include the characteristics mentioned in section 5.1.

- Mailchimp UX Library [mai]
- Yahoo pattern library [yah]
- Patterntap [pat]
- UI-Patterns.com [uip]
- Smiley Cat [smi]

5.3 Related Work

There is not much related work regarding the creation or evaluation of Pattern Libraries. The few works we have found are:

- Implementing a Pattern Library in the Real World: A Yahoo! Case Study [MLW05]
- Bauanleitung für eine Pattern Library (German) [Brü]
- Creating Style Guides [Rob]

The Yahoo! Case Study comes to the conclusion that there is no currently known tool which eases the creation of a pattern library without huge modification needed. We therefore evaluate on what a pattern library has to include and how to build it. The other two works evaluate the creation of a pattern library. All these works may be used as a guideline towards the creation of an own pattern library.

6 Test Applications And Frameworks

In the context of this process analysis we are searching a tool which improves the testing process. Currently there are already tests in place which test the functional behavior of the web application. These functional tests do not cover whether the site complies to the style guide or not. The visual testing is manually done at the moment.

This section describes different frameworks which can be used to automate the visual testing of web pages. All of the mentioned frameworks are based on the same principle.: They compare screenshots of different revisions of the web pages under test and try to detect any visible differences. The occurring differences have to be checked by a user.

6.1 Features And Requirements

The tools used during the visual testing of the web pages need to fulfil the following criteria:

- **License**: The tool in question should be licensed under an open source license such as the BSD-License or Apache License. Our industry partner mentioned that the openness of the code is very important, because it allows customization and improvement. Furthermore it ensures that there are ways to gain further support in case the original maintainer of the software decides to stop developing it.
- **Integration into the build process**: For some use cases it is required that the tool in question is integrable into the existing build process. This step is essential for the visual tests during the automated build.
- **Representation of the results**: The tool in question needs to show the results of the analysis in a way that the UX-Team and the Developer Team are able to inspect them and comprehend their impact. In addition to a side-to-side comparison, an image representing the differences in both screenshots would be a desirable feature.
- **Collaboration**: It would be a great addition if the tool in question supports and encourages collaboration between the Developer Team and the the UX-Team. Both parties should be able to communicate via this system directly.

6.2 Selenium

Selenium [sel] is a set of tools for automating web browsers. It can be used to test web pages and web applications. Its tests are written against the WebDriver-API. There are many different implementations of the WebDriver API offering support for many popular browsers.

There are two different approaches using Selenium. Selenium offers the Selenium IDE which is a very basic tool to record and playback testcases using the WebDriver. Selenium allows developers to write their tests against the WebDriver-API using different programming languages.

Testing with Selenium is done in two steps. At first the browser input needs to be automated. After that the state of the web page under test is verified. This verification is done by selecting components and then using verifications and assertions on them. Selenium supports different mechanics e.g. using Identifiers, IDs, CSS, XPath and DOM for selecting components.

Selenium by itself is able to create screenshots of a web page under test. For the calculation of the differences between the created screenshots another tool will be needed. ImageMagick [ima] is a tool which can deliver such functionality. Selenium is an appropriate tool because it can be directly called from the commandline or can be invoked by one of the many available programming interfaces in different languages.

6.3 Huxley

Huxley[hux] is an open source tool created by Pete Hunt, a developer at Instagram (Facebook). It offers two different modes to automate GUI tests and detect GUI regressions.

In the record mode Huxley uses the Selenium WebDriver to open a web page and to record all user actions. The user can take screenshots by pressing enter in the Huxley terminal. The combination of user input and generated screenshots is later used to test the GUI for regression.

The testing is done with the playback mode. It reruns the recorded tests with Selenium and takes screenshots to compare them with the existing ones. Differing screenshots have then to be checked by the developer. Afterwards the developer has the possibility to fix his code to prevent the GUI from being changed or approve the screenshots and commit them to the revision control. A designer has the chance to review the changes to the GUI directly in the revision control system by looking at the changes of the screenshots.

6.4 Wraith

Wraith[wra] is a screenshot-based testing tool created by developers at BBC News. It uses either PhantomJS or SlimerJS to create screenshots of different web page environments. After that, it marks differences in the images.

Wraith wants to satisfy the need to test dynamic content. The approach of Wraith is based on the assumption that the local development system is accessing the same data source (e.g. database) as the baseline system. With this approach each of the systems will display the same dynamic content.

Screenshots created by Wraith need to be kept and managed by hand - Wraith doesn't provide any support for this. Wraith can be extended by *wraith-donk*[don] which offers the possibility to wrap Wraith inside a web browser and even allows to send emails containing the results of a test run.

6.5 Depicted - Dpxdt

Depicted-dpxdt [dep] (hereinafter referred to as Depicted) uses a similar approach to Huxley. It establishes a baseline release with an initial set of screenshots of the web page which are later used to do regression tests.

Depicted defines a process which allows external stakeholders (from the developers point of view) like an UX-Team to participate in the review process of changed user interfaces.

The following steps describe an usual approach:

1. Use a baseline release to create an initial set of screenshots.
2. Run Depicted on your current release to create a new set of screenshots and automatically mark the differences of baseline and current release screenshots.
3. Use the Depicted web interface to manually approve or reject each difference found by the tool.
4. Finally mark the release as good or bad.
5. The approved release will become the baseline for future runs.

Dpdxt comes with some handy tools to automate the generation of the screenshots. They offer functionality such as:

- Crawl a website from an URL by a given depth and create screenshots for each of the pages visited.
- Compare two URLs side-to-side.

- Create screenshots for a set of URLs specified in a config file. This also allows the injection of Cascading Style Sheets (CSS) and JavaScript (JS).
- Find differences in already existing screenshots and integrate them into the web interface.

7 Proposals For Modifications

In this section we describe the possible enhancements and modifications. First we describe how it is important to build an own pattern library. The we show the possibilities for automated testing and propose two approaches. Afterwards we propose how to integrate the former approaches into the current process. Last but not least we describe describe the recommendation.

7.1 Pattern Libraries

As the papers and guidelines in section 5.3 showed, there are no known tools for the creation of a pattern library. During our research we did not find an out-of-the-box solution either and therefore recommend the build of an own pattern library which fulfils the criteria mentioned in section 5.1.

7.1.1 Domain Specific Requirements

Besides the characteristics of a good pattern library described in section 5.1 the tools in question need to support the following features:

- **Collaboration**: The tool in question should support collaboration among the Developer Team as well as for the UX-Team. It should support the creation of different user roles and user groups and should include a at least basic permission control system. The support of further features, such as comments, notifications, tags, etc. is desirable, but not mandatory.
- **License**: The license requirements are identical to the ones described in section 6.1. Briefly speaking this means that the source of the pattern library needs to be available to ensure its future-proofness.
- **Language and Library Support**: The pattern library should be able to support several programming languages and libraries such as:
  - Textual descriptions
  - HyperText Markup Language (HTML)

  - Cascading Style Sheets (CSS)
  - JavaScript (JS)
  - Support for pictures / screenshots
  - Support for displaying code snippets
  - Support for LESS
- **Version Control**: The pattern library should support any kind of version control, to be able to track as well as revert changes. The changes done to the pattern library should be relatable to single users and single commits. If the pattern library itself does not support any kind of version control, it should be usable inside an own version control system. Therefore the stored files have to be in an open, human readable format, such as XML, JSON, etc.
- **In-house Hosting**: As the pattern library contains sensitive data, it is mandatory that the tool in question will be deployed and maintained in-house. Therefore a cloud-hosted solution is not applicable.

7.1.2 Conclusion

To fulfil all the criteria mentioned earlier (general characteristics as well as domain specific requirements) it is inevitable to create an own pattern library. This ensures that all requirements will be met. The section 7.3 describes how the development of a pattern library can be done and included in the current development process.

7.2 Testing Tools

After evaluating the different testing tools we came to the conclusion that the preferred tool depends on the desired usage and the time commitment to the tool.

7.2.1 Selenium

Selenium can be used to test almost everything on a web page, but the test cases need to be tailored for every specific site. Using Selenium can be compared to creating a unit test. Tests need to be updated for every (maybe even small) change in the GUI.

7.2.2 Huxley

Huxley is based on Selenium but uses a quite different approach. The effort needed to create a working test case is quite minimal. The user does the required actions once and defines when a screenshot is necessary. For running the test Huxley just replays those actions and redoes the screenshots for comparison.

Unfortunately Huxley is not that polished and robust, therefore we do not advice to install it on a build server or use it for some high level of automated testing. It may even not be appropriate to use it on a smaller scale, like a developer testing his local system against a baseline system, because it has some major issues (e.g. stopping after the first occurring difference). With a little tinkering it may reap great benefit to use it locally as a developer because of its painless creation and recreation of test cases.

7.2.3 Wraith

Wraith uses, unlike Selenium and Huxley, a headless browser. If extended by *Wraith-Donk* Wraith offers, additional to the already mentioned features, web access to the results. This is a mandatory feature for the automated testing of the build.

A thing to consider is that Wraith needs two different URLs at the same time to create its screenshots and detect differences. This may be a problem depending on the use of this tool.

7.2.4 Depicted

Depicted has almost the same features as Wraith but uses the WebDriver API instead of a headless browser. It is also very comfortable to manage the screenshots, because a passed test is automatically the new baseline. Contrary to Wraith, Depicted generates its screenshots on the same resource. This means it does not detect changes between two builds but also changes on the same resource and can therefore be used to automatically detect changes in the GUI.

7.2.5 Conclusion

Selenium may be used but the time investment may be huge for the outcome. This is not about the initial setup of the tests but also about keeping them synchronized with changes in the GUI of the application.

For automated testing there are two approaches which could even be combined:

Developers Running Smoke Tests On Their Local Machines

A web-developer often opens up his browser to check his changes before committing his code. In this manual test he often fails to spot some (even quite obvious) erroneous behaviour. These unnoticed changes could be spotted by screenshot comparison.

The tests described in this section should only be seen as an addition to the developer manually checking the impact of the changes before committing them.

An important factor for those tests is that they have to run fast if the developer has to invoke them manually. Long running tests may tend to hinder developers from committing and continuing their work or mislead them into not executing the tests at all.

For this use case it would be feasible to use *Huxley* working on a local Selenium Standalone Server to compare the screenshots before and after applying changes. There won't be a need to keep the screenshots under revision control because they will and have to be generated after each checkout and before every commit. The only thing worth keeping under revision control is the configuration file which defines the test steps. The main benefit of this is that the screenshots do not become outdated because they will be generated after each checkout.

Alternatively *Wraith* could be used for this purpose. The only restriction is that, instead of running the tool before and after applying changes to the code, the developer needs a baseline web page running on some server. For this use case a really simple web server will be sufficient (e.g. SimpleHTTPServer included in Python, which can be run directly from the command line). This may even be the better solution than Huxley because during our tests it seemed to be more stable.

This testing approach requires the developers to deal responsibly with changes to the GUI because they decide which changes make it into the code. This situation is quite similar to the situation without this kind of test, but the changes do not get unrecognized into the code anymore.

Automated Testing Of The Build

During an automated build the current state of the application is built and therefore verified in regular periodic cycles. During this process it would be very easy to, additional to other tests, test the GUI for regressions.

The results of the test will show all changes done to the GUI since the last build. For every change a developer has to decide if a specific change is a desired one. In many cases changes will be intended modifications, in this context we could speak of *false-positives*.
These false-positives represent a great expense but can not be eliminated automatically. After each build the new baseline for the next test needs to be defined by accepting or declining occuring differences between the builds.

These tests don't affect developers directly because they only depend on the generation of the build. The generation of a build usually takes a longer time or even runs over night, therefore the tests executed after the build generation may take a longer time in comparison to the tests done by each developer.

Among the introduced test-applications, *Depicted* is the most fitting tool for this approach. Even if Wraith with the additon of Wraith-Honk may fit too, but Depicted looks better at supporting the developers sticking to the screenshot validation process. *Depicted* delivers many reasons to use it in an automated build environment:

- All operations are triggered from a command line, and therefore are perfect for being executed after a finished build.
- Depicted offers a web application for easy and clear access to the results and their evaluation.
- Depicted supports the developers in using a clear defined process to check the occuring changes.
- The approved new screenshots will automatically be used as the new baseline for future tests.

7.3 Process

The following sections discuss proposals for integrating the development process of a pattern library and an automated test cylce into the current development process.

7.3.1 Not Working Proposal

In the beginning of the process analysis we asked why the UX-Team does not provide a programmatically usable style guide. The answer was that the UX-Team is just dedicated to graphics design and is not capable of writing CSS, HTML and JavaScript. Thus the proposal of moving the competences of writing the code to the UX-Team is no suggestion which would be practical at the moment.

However this thought is picked up in the subsection 7.3.2 in which one possible way is to send one of the Developer Team to the UX-Team. This would provide the knowledge needed for writing the code.

7.3.2 Development Of A Pattern Library

The needed improvement of the style guide, to a version which is more convenient for the Developer Team, is claimed in section 4.1. In this subsection we show two ways of integrating the development of the pattern library into the process.

We identified two possible ways of creating a pattern library which focuses on components and the implementation. Both need at least one capable developer who develops the pattern library and may be part of the Developer Team. Also there is the possibility to declare someone to be responsible for the creation and maintenance of the pattern library.

Developer Team Develops The Pattern Library

In Figure 7.1 the developer or developers of the pattern library are and remain part of the Developer Team. They interpret the style guide to create the pattern library. This can be done parallel to the development phase of a sprint.

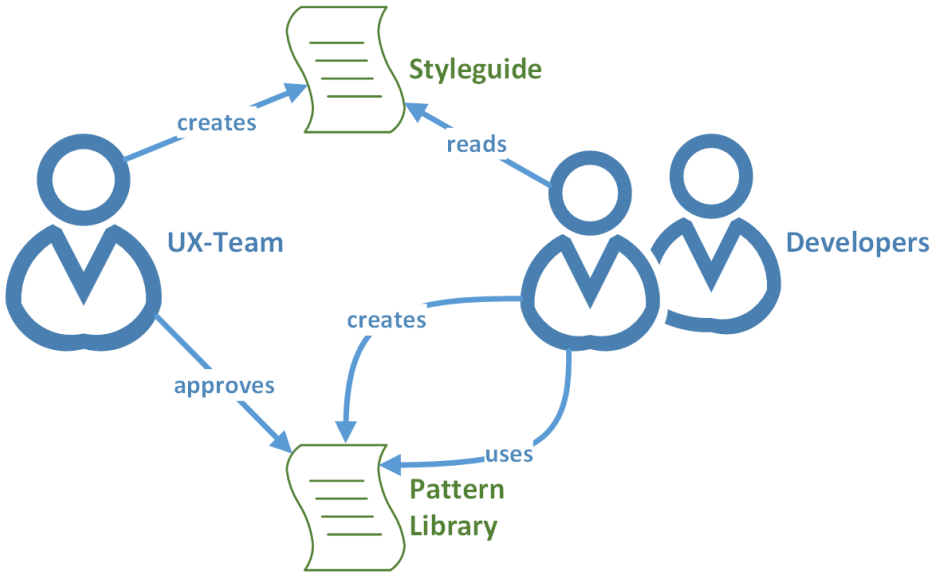

Figure 7.1: Developer Team Develops The Pattern Library

After the development phase or parallel to the testing phase of a Sprint there has to be the validation and the approvement of the pattern library by the UX-Team. Advantage of waiting with the validation after the development phase is to not interfere with components under development. Advantage of the overlapping validation is the possibility to react to defects during the same development phase and more time for the UX-Team.

UX-Team Develops The Pattern Library

Another possibility of creating the style guide according to the subsection 7.3.1 is to delegate one or more developers of the Developer Team to the UX-Team. As shown in Figure 7.2 this would bridge the location gap between both teams.

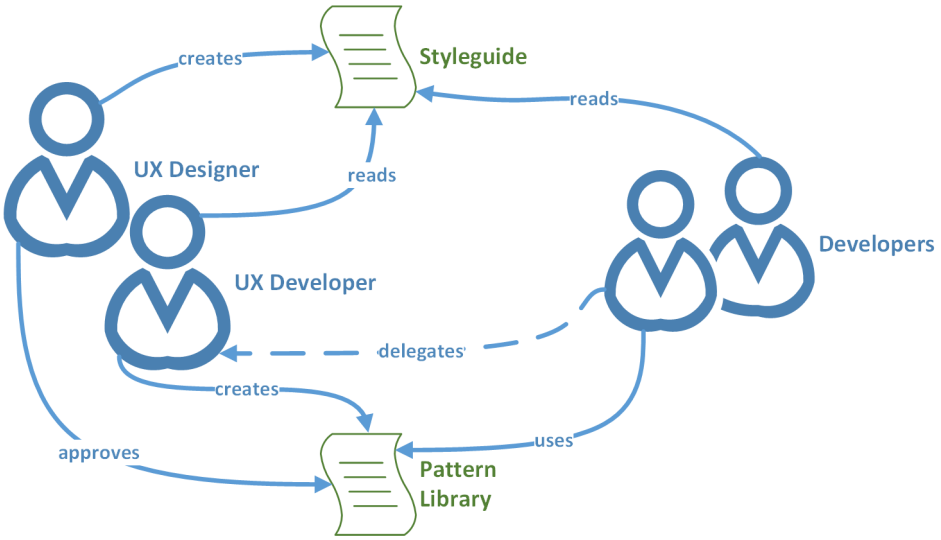

Figure 7.2: UX-Team Develops The Pattern Library

Advantage of this possibility compared to the suggestion above is to decouple the creation and approvement of the pattern library from the development phase of the web application. On the other hand this would mean that at least one of the Developer Team has to be assigned to another department. This would likely result in moving abroad.

Style Guide References The Pattern Library

Both solutions do not alter the need for interpreting the style guide by the Developer Team, nor the problematic format of the style guide and the duplicated information in it. On the other hand the development of the pattern library shows the duplicated information in the style guide and supports the possibility of using the pattern library by the UX-Team. In the best case the style guide is replaced by the pattern library which is unlikely to happen in the near future. But to use the component structure of the pattern library and referencing the components of the pattern library inside the style guide would solve the information duplication and its negative consequences.

7.3.3 Integration Of An Automated Test Cycle

Section 7.2.5 presented a solution to technically integrate an automated GUI Test in an existing build server driven developing process. Besides the raw technical integration, the integration in the existing process needs to be kept in mind too.

There are quite some differences regarding those two aspects of integration. For example, technically it would be perfectly fine and possible to include an GUI Test in every generated build. Regarding the process this would not be very beneficial to the workflow of the developers. The developers would have to check and approve all GUI differences after every build. Checking those GUI differences has quite some overhead work attached, e.g., opening the image comparison tool and setting up a comparison view. Considering that many found errors are false-positives (wanted changes), inspecting all changes during a week at once would prevent an immense amount of additional work.

7.4 Result

As the UX-Team is not able to provide a pattern library and delegating a developer most likely will be too expensive, we recommend that the Developer Team creates their own pattern library. The development of the pattern library should be integrated into the existing Scrum process.

For the automated testing of the GUI we recommend two different tools for the two possible applications. For the smoke tests on the developer machines we recommend either using Wraith or Huxley. For the automated testing we recommend using Depicted. The automated testing should be done after the integration day to react on the testing results during the next development week.

8 Reflection

During the project we gained experience in working with a big industry partner. This unveiled several challenges due to the number of people involved at different locations. This included the scheduling of appointments and communication mainly over distance. Futhermore, due to the companies policy, we experienced bureaucracy concerning access and sharing of information. This resulted in a delay in the process analysis' project schedule. However this did not affect the project's deadline, as we included a big enough time buffer in our project schedule.

Due to the grown structures of the industry partner as well as its enormous experience in the field of software development, the inspected processes were well matured and therefore did not leave much room for improvement. Also the industry partner already did a fair amount of research in order to find appropriate solutions to the issues regarding the style guide, the automated GUI-Tests and the corresponding processes. Due to these circumstances we did not have to completely overhaul the processes in question, but rather than find small enhancements to already well defined processes.

We also faced the problem of working concurrently for different projects and commitments like this process analysis as well as modules of our curriculum. This enforces to elaborate a project plan, dealing with a large communication overhead as well as dealing with other people having simply no time.

To put in a nutshell, the process analysis was a great experience to us. It was interesting to see how a big company develops software compared to our education in managing software projects. Also it was pleasant for us that our mainly theoretical knowledge was applicable to the industry partners processes and enabled us to contribute to the enhancement of these.

9 Summary

During the process analysis we investigated the existing processes of the industry partner through interviews and email conversations. We found starting points for enhancing the current development process. On the one hand the non-technical format and context of the existing style guide is an issue, because it is neither unambiguous nor consistent in itself. On the other hand testing of the GUI appearance is done very sporadically and by hand.

After analyzing these issues we discussed multiple approaches and searched for possible tools to support the Developer Team in solving these issues. Hence a new and developer oriented style guide format has to be created which is commonly referred to as a pattern library. Unfortunately there do not seem to exist any tools to assist in the creation of a pattern library. Therefore we evaluated the manual creation of an own pattern library with investigating existing Pattern Libraries as well as publications regarding the creation of Pattern Libraries.

To introduce the automated testing of the visual appearance we searched for several possible web testing solutions. We analyzed four tools in depth to see wether they fit the requirements. Finally we came to the conclusion that, depending on the concrete task, either Wraith, Huxley or Depicted should be used. Wraith or Huxley can be used for smoke tests, while Depicted can be applied to the existing test process.

After evaluating the creation of a pattern library as well as the available tool support, we tried to adapt the previously mentioned possibilities to the existing process. We found out that our approaches do well fit into the currently running Scrum process.

Bibliography

[Brü] Brüning. Bauanleitung für eine Pattern Library. http://www.produktbezogen.de/bauanleitung-pattern-library-1/. 08/27/2014. (Cited on page 26)

[dep] Depicted—dpxdt. https://github.com/bslatkin/dpxdt. 08/27/2014. (Cited on page 29)

[don] wraith-donk. https://github.com/guardian/wraith-donk. 08/27/2014. (Cited on page 29)

[hux] Huxley. https://github.com/facebook/huxley. 08/27/2014. (Cited on page 28)

[ima] ImageMagick. http://www.imagemagick.org/script/index.php. 08/27/2014. (Cited on page 28)

[mai] MailChimp Pattern Library. http://ux.mailchimp.com/patterns. 08/27/2014. (Cited on page 26)

[MLW05] E. Malone, M. Leacock, C. Wheeler. Implementing a Pattern Library in the Real World: A Yahoo! Case Study. *ASIS&T IA Summit*, 2005. (Cited on pages 25 and 26)

[pat] Pattern Tap. http://patterntap.com/. 08/27/2014. (Cited on page 26)

[Rob] Robertson. Creating Style Guides. http://alistapart.com/article/creating-style-guides. 08/27/2014. (Cited on page 26)

[sel] Selenium. http://docs.seleniumhq.org/. 08/27/2014. (Cited on page 28)

[smi] Smiley Cat. http://www.smileycat.com/design_elements/. 08/27/2014. (Cited on page 26)

[uip] UI-Patterns.com. http://ui-patterns.com/. 08/27/2014. (Cited on page 26)

[wra] Wraith. https://github.com/BBC-News/wraith. 08/27/2014. (Cited on page 29)

[yah] Yahoo Design Pattern Library. https://developer.yahoo.com/ypatterns/about/. 08/27/2014. (Cited on page 26)

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature

---

place, date, signature

---

place, date, signature