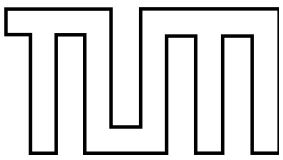


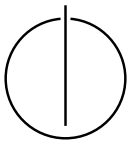
DEPARTMENT OF INFORMATICS

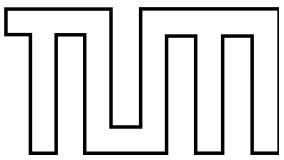
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Do Multi-Fidelity Levels improve Mockup-Driven Development?

René Milzarek





DEPARTMENT OF INFORMATICS

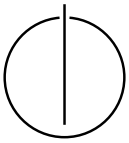
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Do Multi-Fidelity Levels improve Mockup-Driven Development?

Verbessern Multiple Detailgrade das Mockup-Driven Development?

Author:	René Milzarek
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	M.Sc. Adrian Hernandez-Mendez
Submission Date:	15th of December 2016



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15th of December 2016 René Milzarek

Acknowledgments

First and foremost I want to thank my family for always being there for me and helping me in various ways. A very special thanks is due to my girlfriend Franziska, who had to step back over the last weeks, but gave me the room and her understanding to complete this thesis. Furthermore her strength to overcome inhuman challenges and her will of life is inspiring to experience every day.

Secondly I want to thank my Mentor Xaver Zerreis, who gave me the possibility for the industry cooperation, for the trust and freedom I received. Without this it would not have been possible to create the thesis in this form. Especially the conversations, discussions and idea exchanges at the beginning set a solid base for the further research.

Furthermore I also want to thank my colleagues, who supported me with their feedback and their time to participate in the interview and evaluation of the concept. Especially Dominik Speicher always had an open mind for the discussion of new ideas and gave valuable hints and tips for the implementation of the prototypical component catalog.

A big thanks also goes to my advisor Adrian Hernandez-Mendez, who guided the academic research, for the stimulating discussions and the constant suggestion of new ideas and view points. He helped me with his guidance to put the academic research on a broader basis, whilst not losing the relation to the industry partners problem. Furthermore I also want to thank Prof. Dr. Florian Matthes for the possibility to create this thesis in cooperation with Siemens.

Finally I want thank all my friends for their feedback, their proofreading and also understanding over the last weeks. You are appreciated, although you are not all mentioned by name. Thank you!

Abstract

In dieser Arbeit wird anhand einer umfangreichen Literaturrecherche eine grundlegende Definition für den Begriff des Detailgrades bei Prototypen und für den Mockup-Driven Development Prozess gefunden. Hierbei wurden zwei hauptsächlich verwendete Detailgrade identifiziert und weitverbreitete Usability Artefakte dementsprechend klassifiziert. Des Weiteren wurde der Entwurf von Prototypen mit multiplen Detailgraden als der gezielte Prozess der Erhöhung des Detailgrades durch das Hinzufügen neuer Designeigenschaften definiert. Der Mockup-Driven Development Prozess beschäftigt sich ausschließlich mit dem Übergang von Prototypen mit einem hohem Detailgrad zum eigentliche Produkt und entspricht somit nicht dem zuvor beschriebenen kontinuierlichen Prozess der Detailgraderhöhung. Ein weiterer Mockup-Driven Development Ansatz, welcher als *Cascading Tree Sheets* bezeichnet wird, erwies sich als mehr oder weniger irrelevant für das Themengebiet der Prototypen-Entwicklung.

Eine Anforderungserhebung beim Industriepartner Siemens identifizierte die Verbesserung der Zusammenarbeit und die Ermöglichung einer systematischen Wiederverwendung von Komponenten als Hauptkriterien an einen nutzerzentrierten Entwicklungsprozess für Prototypen. Auf Basis dieser Anforderungen wurde ein solcher Prozess entworfen und gemäß der eingeführten Begriffsdefinitionen als Prototype-Driven Development Prozess bezeichnet.

Der Ausschnitt der Komponentenverwaltung und -erstellung aus dem definierten Prozess wurde mithilfe einer prototypischen Implementierung eines Komponentenkataloges umgesetzt. Das Gesamtkonzept wurde im Rahmen eines Usability Walkthroughs und eines nachgelagerten Fragebogen mit Mitarbeitern bewertet. Der vorgeschlagene Prozess trug zu einer Erhöhung der Zusammenarbeit und Wiederverwendung existierenden Komponente bei, jedoch konnte ein Mehrwert bei der Unterstützung multipler Detailgrade nicht festgestellt werden.

Contents

Part I.

Introduction and Problem Identification

1. Introduction

1.1. Motivation

Mobile devices accounted for 60% of the media time spent by U.S. users in 2014, which reflects an absolute increase of 9% to the previous year (see figure 1.1). The usability of apps in a more and more competitive market is increasingly becoming a key differentiator in the Business-to-Consumer (B2C) market. According to Gartner this trend also affects businesses and results in an increasing "demand for apps in the enterprise [which are expected to] meet the high performance and usability of consumer apps" [52]. Another term used for this phenomenon is the “Consumerization” of enterprise IT and vividly describes that employees expect the same user experience from enterprise software as they are experiencing from the software used in their private life.

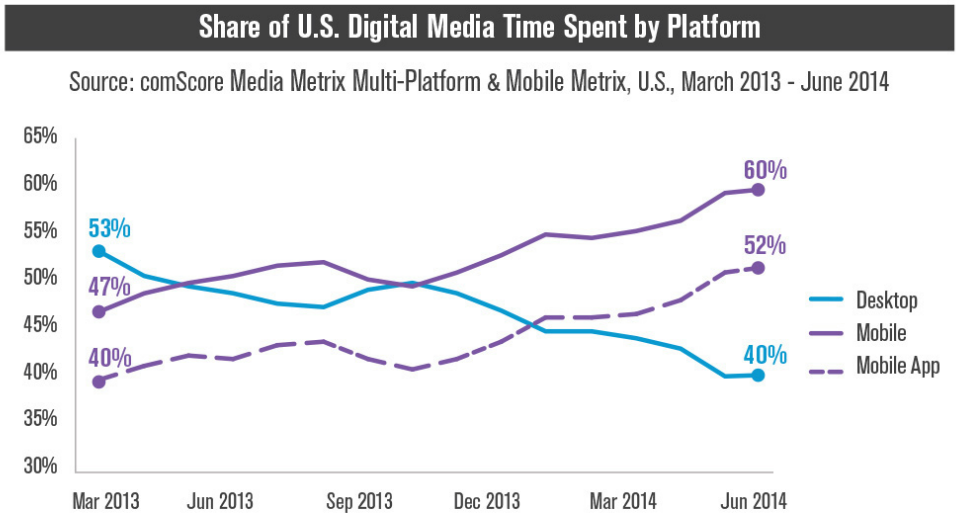


Figure 1.1.: Share of U.S. digital media time spent by platform [20].

Instead of taking a defensive position against this trend, enterprise IT should embrace it as a chance "[...] to introduce new business efficiencies and innovations and [to] increase ITs relevance to employees" [51]. This could be achieved by the introduction of User-Centered Design (UCD) methods like personas, prototyping or usability interviews and questionnaires. These techniques are not only valuable to support internal software development projects, but also during the evaluation of Commercial Off-the-Shelf (COTS) solutions from external vendors [51].

The enduring discussion about the Return on Investment (ROI) of usability engineering from a qualitative as well as from a quantitative perspective [18, 30, 26, 16, 5, 66] combined with the expectations of the end users seems to have had an impact on decision makers in enterprises. Rather then thinking about the if, companies are actively looking for solutions on how to implement and integrate a user-centric approach in the existing development environments.

Especially in the case of internal software development projects the budget is limited and does often not account for the involvement of a dedicated usability expert. It should be noted, that this is not valid for the core business, where UCD is an essential aspect and external agencies are often supporting the product development. Internally the activities of the usability engineering process are often spread over multiple persons and roles. The analysis phase of the usability process is mainly supported by the requirements engineer whereas the software engineer implements activities of design phase [14]. This already indicates that there might be challenges in the collaboration between these roles. Depending on the established process the stakeholders are working on the same artefacts, but potentially on different degrees of fidelity. A potential solution for the transition from mockups to code was proposed with the mockup-driven development approach [4, 46]. The idea of an efficient and integrated UCD process is a key factor for the acceptance in enterprises and environments were the development budget is very limited.

1.2. The Cooperation Partner and Scope of Application

A big part of the research was motivated by and executed in cooperation with the industry partner Siemens AG. The company had 351.000 employees worldwide in the fourth quarter of 2016, which of 113.000 were located in Germany [49]. Concretely the cooperation was established with the central IT department for human resource and supply chain management applications. The department implements IT demands by buying and adapting Commercial Off-the-Shelf (COTS) solutions from vendors or the internal development of individual solution in the Java competence center.

1.2.1. Software Development Process

The department's software development process is divided into 3 phases – the start, the initialisation and the implementation phase – and is followed by a transport to the test system (see figure 1.2). After the tests were executed positively another transport to the production system is performed. The start phase's main focus is to ideate and identify reusable components, which could be added to the department's "*Basic Component*" catalog. The initialisation phase takes care about the organisational and infrastructural aspects of a new software project. This, for example, includes granting access to the development systems and project repository as well as the setup of a build plan on the internal Continuous Integration (CI) server. The implementation phase does not only involve the implementation of source code, but also the configuration and integration of standard services like the authorisation and role management service.

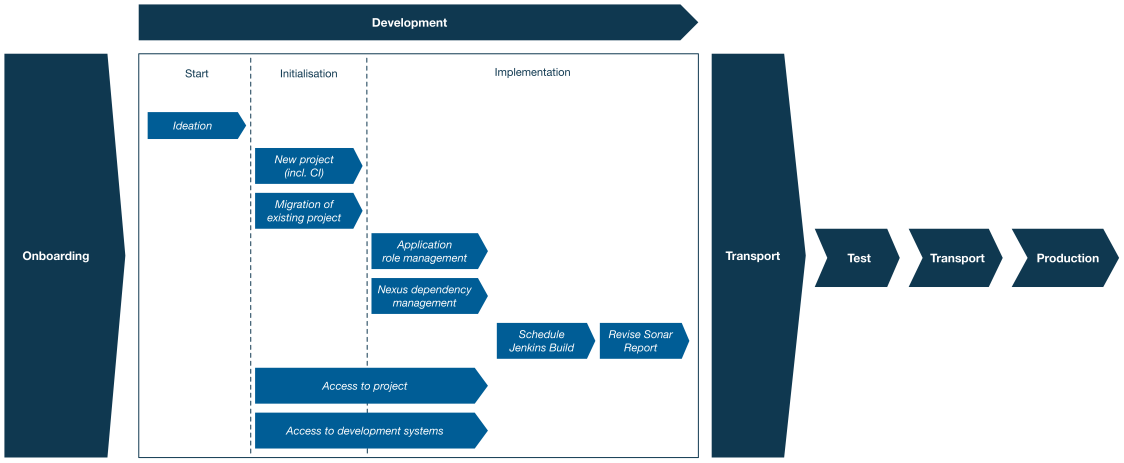


Figure 1.2.: Software development process of the industry partner's engineering department.

Currently, the software development process is heavily oriented around technological aspects. Especially the setup of the development stack for new employees and external consultants is introduced. User-centered design methods are not mentioned in the development guidelines, which does not reflect the actively practiced process. Several activities of the usability engineering methodology are indeed regularly used and the collected feedback impacts design decisions. The current situation is going to be examined in greater detail in chapter 2 as part of a semi-structured interview with experts.

1.2.2. Consumerization Trend

Siemens also experiences the previously described trend of a growing mobile user base, which comes along with an increasing demand for usable mobile applications and a

stronger awareness for usability in general. Since the start of the company's Mobile Device Management (MDM) service in July of 2015 the amount of enrolled devices increased to a total amount of 30.831 in November 2016. This reflects a growth of 17.3% during the last month. The overall number of mobile devices was 161.000 in November 2016, which comprises all devices with access to the corporate email servers and is a superset of the MDM enrolled devices (see figure 1.3). Supported by the rollout of a company internal social network, the users are more empowered and actively express their opinions and requirements through the new communication channel. This comes with the benefit of a faster communication, broader reach and thus higher probably of reaching a responsible colleague, who is able to consider the feedback for a future release.

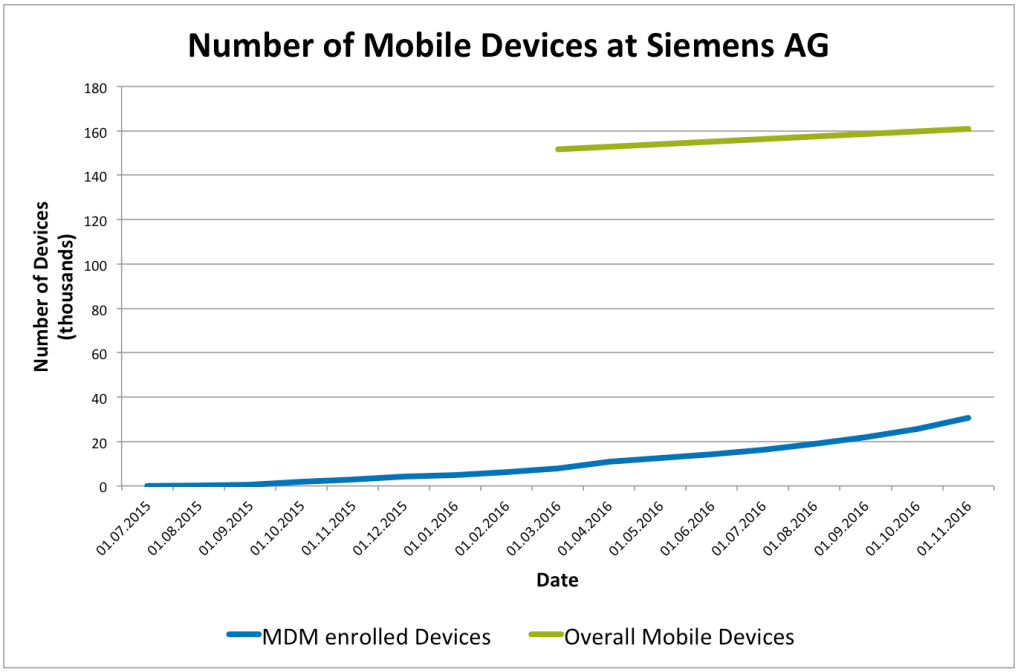


Figure 1.3.: Number of mobile devices at Siemens AG.

1.2.3. State of Usability Engineering Adoption

The efforts to more intensively include the feedback of the end users started already in 2008 with an open usability survey encompassing the whole HR tool landscape. The results were 729 proposals for improvement from 139 employees and motivated my bachelor's thesis "Analysis of the added value of innovative usability and dynamic visualization concepts for the graphical representation of Siemens AGs KPIs and prototypical implementation of a native iOS-application", which addressed the demand for supporting approval processes on mobile devices [37]. In February of 2015 another

initiative was started in the form of a survey in the company's internal social network. The goal of the survey was to identify future mobile app use cases, which could simplify the work life of Siemens' employees. Within 7 days 200 ideas and more then 2.200 votes were collected. A top ten list was extracted of which 4 ideas addressed apps in the area of communication and real time collaboration. Another 4 ideas addressed topics, which are located in the direct responsibility of the industry partner's department.

One request was the ability to access the employee's contact details via mobile devices (especially from Android devices, as a solution for iOS already existed). A concept for this use case was developed in the form of paper-based sketches and is going to be presented in section 4.2.1 as part of the solution design to evaluate the suitability of the prototyping processes. Another example for the current utilization of usability engineering was the extension of a web application by a new module for the management of the employee's variable compensation (see section 4.2.2). The user interface was conceptualised with the prototyping tool Justinmind in the form of high-fidelity mockups.

1.3. Research Methodology and Organisation

The thesis follows the "Design Science" research approach, which uses the creation and evaluation of IT artefacts as a mean of solving existing, organisational problems [17]. This approach focuses on the close cooperation between researchers and practitioners and is in the very nature of this thesis as it is motivated by an industrial application. Nonetheless, Offermann's research process, consisting of the three phases of problem identification, solution design and evaluation, was utilised to ensure a correct, comprehensible and transparent approach during all phases of this research.

In chapter 2 and following this approach, the problem and its relevance for the practical application is presented. Furthermore, the research questions, which are guiding the rest of the thesis, are introduced. Therefore, informal interviews with the department's head, the head of development and a requirements engineer were conducted to get a basic understanding of the problem domain and gather Siemens' requirements. Afterwards a guide for a semi-structured interview was developed to support a second round of interviews with 11 employees to validate the previously collected requirements. The roles of the employees were spanning from demand manager, requirements engineer, software tester, software engineer, software architect to project lead.

Afterwards, in part II, the solution design is developed. The part is divided into 3 chapters. The first chapter 4 introduces the mockup-driven development process and compares it to other prototyping processes. Subsequently, two corporate development projects, which applied UCD methods, are presented and assessed on their degree of adherence to the presented processes and on their suitability for adaption to the pro-

totyping processes. In section 3.2, the term multi-fidelity is analysed and defined on the basis of a profound literature research. Afterwards the connection between the mockup-driven development process and multi-fidelity prototypes is illustrated and a transition to the implementation of a prototypical solution is made in chapter 5.

Finally, in part III the added value of a multi-fidelity component catalog is evaluated. Therefore, a usability study was carried out with a total of 8 participating employees. The overall usability of the suggested prototyping process (including the developed web application) was examined with the System Usability Scale (SUS) questionnaire. After summarising the results, issues and ideas for future work (e.g. further integration potentials of the web application) conclude the thesis.

2. Problem Identification

As mentioned in the introduction this thesis was motivated by a concrete problem of the industry partner's department. Section 1.2.2 and 1.2.3 have already briefly described the situation, which is going to be analysed in greater detail in this chapter.

The department was actively looking for a software solution allowing them to integrate the UCD approach into their software development process. The management had several requirements mainly targeting a software tool, which were collected in an informal interview and summarised in the form of a slide deck. Section 2.1 is going to introduce and explain the requirements. In the next step a semi-structured interview with several experts of varying roles was conducted to evaluate, if the collected requirements are also relevant for the employees. The results of the second interview are going to be presented in section 2.2. Finally, an analysis of existing prototyping tools with respect to their suitability to fulfil the specified requirements was made.

2.1. Requirements Elicitation

The informal interview for the first collection of requirements was conducted with the head of the department, followed by a second interview with the head of development. The motivation of the industry partner was to design a solution, which improves the integration of the usability engineering methodology into their software development process. The expectation was to be able to stronger involve the user (e.g. by designing prototypes and collecting user feedback), while not generating a huge amount of additional work. The focus was placed on the systematic reuse of user interface prototypes for the generation or scaffolding of user interface code. The current practice of designing mockups and solely using them as a requirements specification for the software engineers should be avoided.

Below the requirements are listed by their priority and shortly described in the following section.

1. Collaboration
2. Shared and reusable component catalog
3. Export or generate code for the UI

4. Integration with the Application Lifecycle Management (ALM) software
5. On-premise solution
6. Test on target platform
7. Platform support of the prototypes

The support and enhancement of collaboration between the stakeholders of the development process was stated as the most important feature. This rating could be explained by the internal structure of the department. There are individual teams for the different functions: demand management, requirements engineering, software testing and software development. Furthermore the spatial separation of the software development team, which is located in Paderborn, from the remaining team, which is residing in Munich, even increases the challenge of an intensive collaboration between all stakeholders.

One interface between the requirements engineer and the software engineer is the existing user interface components. During the analysis phase of a new software project they have to identify existing components, which could be reused, and agree on components, which have to be newly created. Internally, the components are called "Basic Components" and are maintained by the software developers. However, this catalog is not linked in any way to the prototyping tool of the requirement engineers. The consequence is, that they have to create another component catalog within the tool and manually assure the consistency.

Thus, a component used within a user interface prototype either has a counterpart in the form of an existing user interface component or its purpose is to specify the creation of new user interface components. The only difference between the representations is, that during the requirements engineering a lower fidelity might be used. The envisioned solution supports a transition from prototype to user interface code. The management emphasised the importance of this idea and wants to avoid the currently existing sharp division.

As described in the introduction, there is an established software development process, which is mapped to the ALM tool. Currently, the usability measures are decoupled from the development process. Not only the UCD process, but also the prototyping tool should be able to integrate with the existing ALM solution or at least offer interfaces, which allow the integration.

The management preferred an on-premise solution, which could be operated in the internal data center. However, the information derivable from a user interface prototype is not confidential, if certain requirements are met. The prototype must not contain

intellectual property or personal data, which is applicable for the projects in most cases. Therefore, this requirement has a lower priority and does not represent an exclusion criterion.

Another requirement was the ability to test the prototypes on the target platform of the respective application. As mentioned in the introduction the increasing demand of mobile application is a trend, which also could be observed internally. Thus the prototyping tool should be able to execute a mobile prototype on a real mobile devices. The importance of this feature is based on several studies, which have shown that the environmental conditions are heavily influencing the collected feedback of the users [13, 3]. The following example should illustrate this issue: Imagine that you are collecting feedback for a mobile application by presenting it to a user on your smartphone. The user criticises that he is not able to hit a specific button as several buttons are placed too close to each other. This kind of feedback could not be collected on a desktop computer as it does not provide the typical touch interface.

The final requirement is directly linked to the previous one and does not address the operating system support of the prototyping tool, but the support of target platforms. It should be possible to create prototypes for web applications as well as for mobile applications. This feature was of minor importance as it is often possible to represent the mobile platforms by importing existing component catalogs.

2.2. Expert Interviews

In a second stage the collected requirements were additionally examined with a semi-structured interview with $N = 11$ experts of different professional and organisational backgrounds. In chapter A of the appendix the comprehensive evaluation of the interview can be found. However, this section will only present the key results. The figures on the following pages follow the convention of depicting the absolute number of persons in round brackets.

The majority (72.7%) of the interviewees were employees of the industry partner, thus worked for a large company. Additionally, one employee of a micro, small and medium company was interviewed (see figure 2.1).

A variety of 6 different roles was covered by the interview assuring the representativity of the results (see figure 2.2). The two requirement engineers also have the roles of software testers, but spent the majority of their work time in the requirement engineer's role. That is why the role of a software tester does not occur in the analysis.

The years of professional experience ranged from 0 to 23 years. The median was 10 and the mean 11.82 years ($\sigma = 7.93$). This proves that the interviewed experts have a

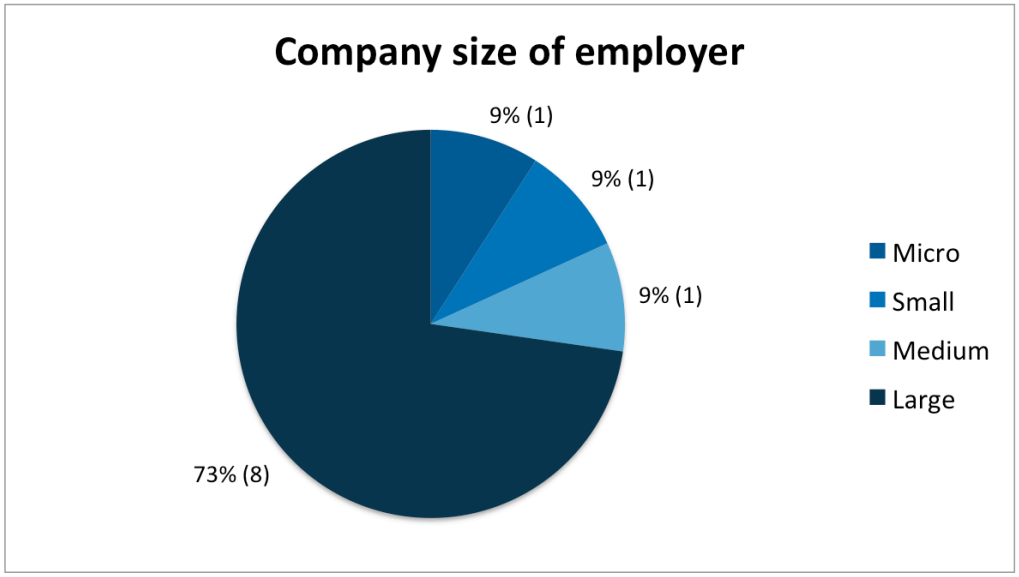


Figure 2.1.: Company size of the interviewee's employer.

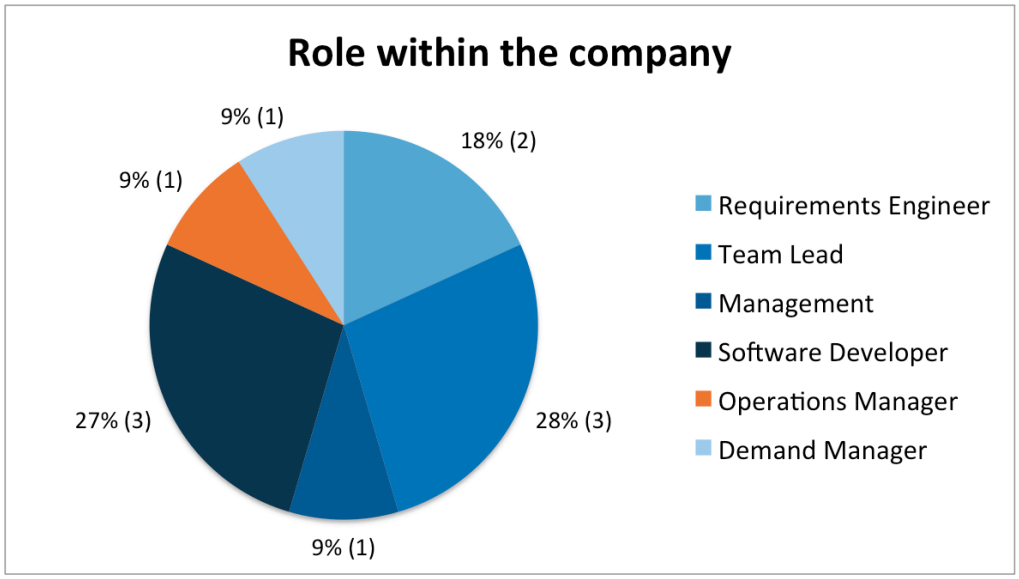


Figure 2.2.: Role of the interviewee within the company.

sound basis of professional experience, which is often considered to be one of the key characteristics of an expert [34].

Almost all interviewees (81.8%) have used UCD methods in their professional lifes. Only two software developers (18.2%) have never actively applied any usability method (see figure 2.3). However they both reported to have received high-fidelity mockups as a specification document. During the evaluation of the personal experience with the user-centered design approach the possible implications of this practice is going to be described in detail.

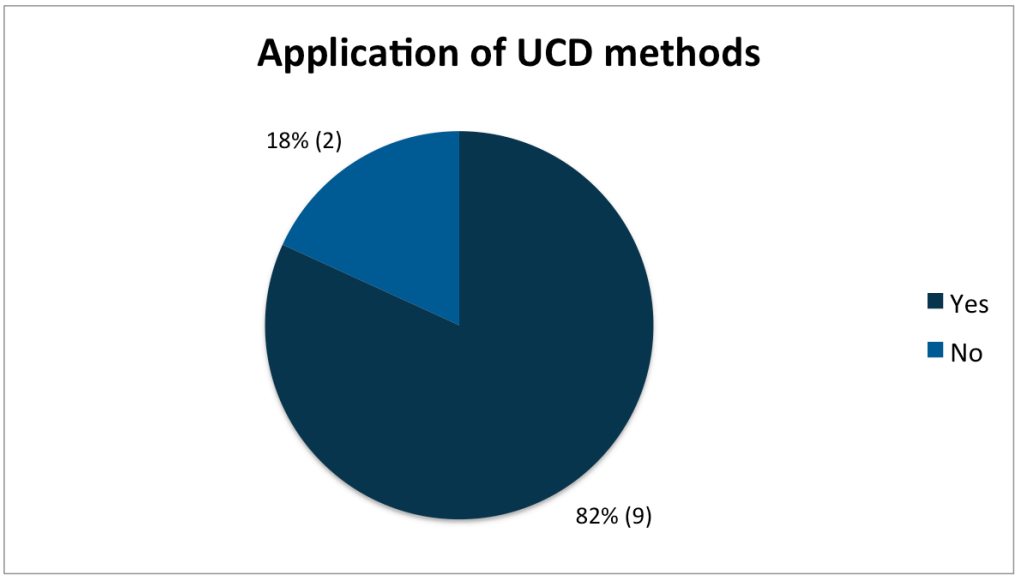


Figure 2.3.: Number of persons applying UCD methods.

The majority of interviewees – namely 9 persons – worked with high-fidelity mockups. Three persons reported to have utilised high-fidelity software prototypes and another 3 persons interviewed users to systematically collect feedback. One person commented that interviews are the preferred elicitation method, if the user's occupation is not IT-related. The reason for that is the possibility to ask additional questions, if the feedback was not clear enough. Surprisingly, the cheap method of paper-based low-fidelity sketches was only used by one interviewee. However, it is not clear, if this method is not applied or if it was valued to not be worth mentioning due to its low-fidelity character. For future interviews in the context of UCD methods one should consider to collect a list of common methods and let the interviewee chose known ones instead of asking an open question.

During the requirement elicitation 10 persons employed UCD methods. Whereas only 5 interviewees used them throughout the whole development phase of the project.

The feedback given indicated that the difference is connected to the software development approach applied in the project. Classical waterfall projects tend to focus the application on the requirement elicitation and rarely during later phases of the project. In contrast agile software development projects utilise UCD method during the whole development phase. This finding is interesting but not surprising as the iterative approach of agile development requires the team to repeatedly cope with new requirements.

Respectively 4 interviewees saw the purpose of applying UCD method in the definition of a shared language between the stakeholders through the discussion as well as the visualisation of the requirements and the better understanding of problem and process at hand. The following reasons were each identified by 2 interviewees. The application of user-centered design methods helps to avoid late changes of requirements, which are – as we are all aware of – connected with exponentially higher costs the later they are introduced. Furthermore the usability artefacts help to examine if the proposed solution actually solves the identified problem. Another aspect was the improvement of internal team communication and the management of expectations. This accounts for the expectations of the own management as well as the expectations of the customer.

In total 9 persons (81.8%) applied 8 different usability engineering methods, but also 9 persons (81.8%) reported to not have an established development process or guideline, which integrates the user-centered design approach. Only one interviewee answered this question positively and explained that there is an established process, which requires the creation of mockups and UI designs for requirements before handing the specification over to the development. One person reported that some methods are mentioned in the company wide project management guidelines, but not in the internal development guidelines. These results will taken up in the course of the discussion at the end of this section.

Finally the personal experience with UCD methods was rated to be predominantly positive by 9 persons. It enables a better understanding of user's needs and the additional effort invested is justified by the benefits generated (both reported by 3 employees each). The methods were perceived as inspiring and creative. Furthermore their application is very simple and enables faster feedback cycles and better planning. However, the weak points and negative experiences are of special importance to this thesis as they represent the room for improvements. Mockups could create the expectation that the user interface is almost finished and ready for production use. This risk of making a false impression is higher for non IT-related stakeholders, as they do not know the technological difference between a mockup and the real UI code. Furthermore this also addresses one pain point of the management, which is the disposable character of prototypes. The aspect of communication and cooperation between the different involved stakeholders was also seen as a challenge. Usually there are boundary conditions,

which have to be considered by the software developers like the technological feasibility or the reuse of existing components. These conditions might be ignored, if the mockup is created independently by a requirements engineer without regular coordination with the development team. This situation is going to be more closely examined in the course of the case studies in section 4. Another interviewee, who started using UCD methods several weeks ago, reported that the lack of a standard component catalog was seen negatively. It would have been useful, especially at the beginning, to know which components could be supported out of the box and how the general style looks like without reading through several hundred pages of the corporate design guidelines. Finally the difference between the customer and the real user is often neglected during the usability interviews. Most of the time the customer is also a user of the application and the only person, who is giving feedback. However, this dual role might create the a risk of biased analysis, which does not reflect the opinion or experience of a real user.

In summary UCD methods are regularly applied by a variety of different roles. Though, the focus heavily lies on the creation of high-fidelity mockups. Despite the wide application and the generally positive feedback UCD is currently not integrated in most of the development processes. This interview confirmed the importance of the top three requirements identified in the informal interview of the previous section. The collaboration between the stakeholders is of major importance. Especially the communication and coordination between the requirements engineers as the creators of the mockups and the software developers as the implementors of the solution is crucial to avoid designing overly complicated mockups and enabling the systematic reuse of existing software components. Although the approach carries the word user its name, one has to ensure that differentiation between the customer and the user of the software is consciously made. The creation of personas, which was not as established as the creation of mockups, could help to raise the awareness for this distinction. Directly connected to the previous point is the creation of a shared component catalog, which provides an orientational guide of the existing development landscape. Usually IT projects do not start on a green field and have to obey to certain boundary conditions. These ideas of a systematic reuse of existing components is remotely linked to the management's request of the ability to export prototypes to UI code, but it was not mentioned concretely during the expert interviews.

2.3. Prototyping Software Comparison

After the confirmation of the basic requirements with the help of the expert interviews, existing prototyping software tools were evaluated (see figure D.1). The checkmark ("✓") indicates that the requirement is fulfilled by the application. The circle ("○") describes that the feature is only partially supported or that it could be supported with an additional effort (e.g. by using available APIs to extend the tool). The cross ("✗") means that the application does not support the requirement at all and the question

mark indicates that there was no information available about the specific feature.

Two prototyping applications stand out with a high degree of requirement fulfilment.

	Justinmind	iRise Studio	Balsamiq	Pixate Studio Beta	Visual Paradigm
Collaboration (deliver to end users, collect feedback)	✓	✓	○	○	○
Custom Component Catalog	✓	✓	✓	✗	?
Export Code	✗	✓	✗	✗	✗
Integration with ALM (link to requirements, single source for reporting)	○	✓	○	✗	?
On-Premise Solution (host collaboration platform internally)	✓	✓	✓	✗	✗
Test on the Target Platform	✓	✓	✗	✓	✗
Platform-Support (create mockups for mobile and desktop application)	✓	✓	✓	✓	✓
Multi-Fidelity Mockups (support transitions between fidelity levels)	✗	✗	○	✗	✗

Table 2.1.: Comparison of prototyping tools according to the specified requirements.

The first solution is Justinmind (`https://www.justinmind.com`) and is currently used within the department. However, it does not offer the possibility to export the created prototypes to user interface code. Furthermore the integration into ALM tools does not come "out of the box", but there is an open plugin API, which could be utilised to extend the software. It should be noted that after this comparison was made, Justin-

mind released their own Atlassian JIRA (https://www.atlassian.com/software/jira) plugin [23]. Furthermore the product page was updated and now claims that an integration into Microsoft TFS and Doors is possible as well.

The second solution - iRise Studio - offers all specified features. It integrates into 9 different ALM solutions and allows the export of prototypes to code with the help of a template engine. The negative aspect of this solution is the price tag, which will be presented in a comparison in the following section.

Unfortunately the vendors did not reply to price requests for their enterprise versions, thus a comparison could only be made with publicly available data. The Justinmind Professional version has a one time cost of 495,00 USD and the price of iRise Studio starts with the Professional version at 6.995,00 USD [9]. In 2004 the price for 10 iRise Studio Enterprise licenses was 250.000,00 USD [24]. However, these prices are probably outdated and do not represent the enterprise versions, which were the basis for the comparison.

In summary there is a solution on the market – iRise Studio –, which fulfils all requirements, but comes with a very high price. As this does not represent an interesting scenario for an academic thesis further possibilities for the enhancement of a tool supported UCD development approach were searched.

2.4. Research Gap Identification

The process of creating prototypes usually starts on a low-fidelity level, e.g. with a sketch on paper, and advances towards more and more rich prototypes till the final user interface is reached. This issue was also identified by Coyette, Kieffer, and Vanderdonckt in [11], but they focused on mainly on the transition from sketches, which they defined as having "no-fidelity", to the first digital representation with a gesture recogniser. The prototyping tools were also examined for the support of a transition between multiple fidelity levels. The result was that only one tool – Balsamiq (https://balsamiq.com/) – partially supports this feature. It is possible to switch between a "Wireframe" and "Mockup" view, which implemented by switching between two stylesheets for the predefined components. Thus the systematic support of a multi-fidelity approach was identified as a research gap.

In [47], [4] and [46] the mockup-driven development process was introduced, which closely adheres to the idea of a systematic reuse of mockups for the generation of code. This paradigm should be integrated into the UCD process to be developed in the course of this thesis. In section 3.3 a comparison of the three approaches is presented as each one has a slightly different focus.

2.5. Research Questions

Finally the in section 2.4 identified research gap lead to the establishment of the following three research questions:

RQ1 What is the definition of mockup-driven development and the different fidelity levels?

RQ2 What are the requirements for a multi-fidelity mockup-driven development system and how could an implementation look like?

RQ3 How to evaluate if a multi-fidelity mockup-driven development system improves the software development process?

Part II.

Solution Design

3. Definition of Terms

To answer **RQ1** (see section 2.5) the existing definitions found in literature are described in this chapter and a definition, which is used within this thesis is formulated.

3.1. Fidelity-Level

The first impression that there has to be a clear definition for the term fidelity in the context of user-centered design has not been confirmed. The different papers and articles analysed in the course of this thesis used a variety of terms and sometimes even in a contradictory way within the same article. There are sketches, wireframes, mockups or prototypes, which might have a low-, medium-, high- or even no-fidelity.

The Oxford dictionary defines the term fidelity as "the degree of exactness with which something is copied or reproduced" [42]. This provides an useful basis for a fidelity definition in the context of usability engineering. Thus the fidelity of a prototype is the degree of which it corresponds to the final user interface. This definition is consistent with the one of Coyette, which defines "the prototype fidelity [as] the similarity between the final user interface [..] and the prototyped UI" [11]. However, this immediately poses the question of how to measure this degree of fidelity. Therefore a basic button element was analysed on different fidelity levels. The first and lowest fidelity representation of a button was a paper-based sketch (figure 3.1). The highest fidelity level was examined on the basis of the two frontend frameworks *Twitter Bootstrap* (figure 3.3) and *Material Design* (figure 3.4). The last representation was created on a "medium fidelity level" with the prototyping tool *Justinmind* (figure 3.2). Then a comparison of the number of style properties was made.

The sketch had 7, the prototype 37, the *Twitter Bootstrap* implementation 42 and the *Angular Material* implementation 71 style properties. Although the identification of the style properties for the paper-based sketch were subjective, the general trend of a growing number of properties with a higher degree of fidelity cannot be denied. The detailed analysis could be found in the appendix B.

In the course of the increasing fidelity different artefacts like sketches, wireframes, mockups or prototypes are utilised. In literature there is no consistent understanding of their fidelity levels and respectively their field of application. One article defines a wireframe as a "low-fidelity blueprint represented by with grey boxes and placeholders

for detailed content", whereas another one describes it as a high-fidelity artefact [40]. This inconsistency of terms, which spans from blogs to scientific research motivated the following thorough analysis (see table 3.1) [58, 40, 21, 57, 10, 36, 12, 55, 56, 25, 33, 28, 63, 62, 27, 67, 48, 31].

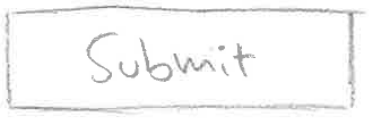


Figure 3.1.: Sketched button

Figure 3.2.: *Justinmind*'s button

Figure 3.3.: Button of *Twitter Bootstrap*

Figure 3.4.: Button of *Material Design*

Analog to the previous table the checkmark ("✓") indicates that the criterion was fulfilled. The circle ("○") describes that the criterion is only partially fulfilled and the cross ("✗") means that it does not apply at all. The criteria were organised in the categories general, fidelity, behaviour, structure, information and style. The last four categories were inspired by the User Interface Modeling Language (UIML), which uses the same structure to model user interfaces and is going to be presented in section5.1 in detail. The results illustrated in table 3.1) are based on a literature research. First a set of relevant articles was collected by searching for the artefact's name and the terms fidelity and definition (e.g. "sketch fidelity definition"). The search was executed on Google's search engine and Google scholar. Afterwards the literature was assessed with regard to the collected criteria. If a new criterion was identified during the analysis the list was extended and a second assessment round was performed afterwards. A full list and the extensive analysis can be found on the enclosed CD.

The common understanding of the examined literature was, that a sketch or wireframe is a low-fidelity prototype, which is often created with pen and paper. Whereas a mockup or software prototype has a high-fidelity and is usually developed on the computer. Furthermore the low-fidelity prototypes merely have any interactive behaviour and only a reduced style. The difference between a software prototype and the real product might not be apparent on the side of the user interface, but rather on the technological side (e.g. the code quality). If you visualise the dimensions of the artefacts with a radar chart one could see that they could be clustered into two groups (see figure 3.5). The low-fidelity group consisting of sketch and wireframe (see figure 3.6), and the high-fidelity group consisting of mockup and software prototype (see figure 3.7).

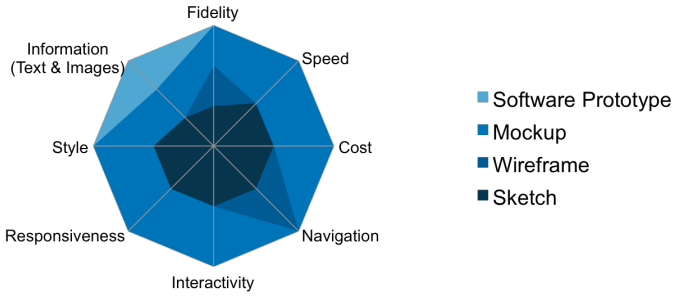


Figure 3.5.: Analysis of UCD artefacts on various dimensions.

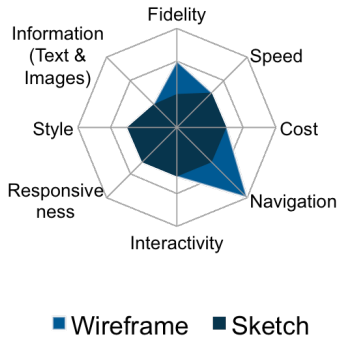


Figure 3.6.: Low-fidelity artefact group.

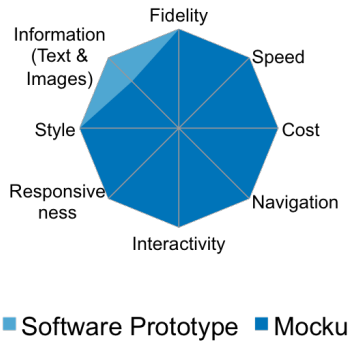


Figure 3.7.: High-fidelity artefact group.

Category	Criterion	Prototype				Product
		Sketch	Wireframe	Mockup	Software Prototype	
General	Technique	paper-based	paper-based	computer-based	computer-based	software-based
	Speed	fast	fast	slow	slow	slowest
	Cost	cheap	cheap	expensive	expensive	most expensive
Fidelity	Low-Fidelity	✓	✓	✗	✗	✗
	Medium-Fidelity	✗	✓	✓	✗	✗
	High-Fidelity	✗	✗	✓	✓	✓
Behaviour	Navigation	✗	✓	✓	✓	✓
	Interactive Elements	✗	✗	✓	✓	✓
Structure	Responsive Design	multiple static screens	multiple static screens	single interactive screen	CSS Other Technology	CSS Other Technology
	Placeholders	✗	✗	○	○	✓
Information	Label	✓	✓	✓	✓	✓
	Text	✗	✗	○	✓	✓
	Images	✗	✗	✓	✓	✓
Style	Colors	✓	✓	✗	✗	✗
	Icons	✗	✓	✓	✓	✓
	Typography	✗	✗	✓	✓	✓

Table 3.1.: Evaluation of fidelity level criteria.

In conclusion this research defines the fidelity of a prototype as the average number of properties used to describe the individual components. A sketch and a wireframe have a low fidelity while the mockup and software prototype have a high fidelity. The medium fidelity level is omitted as it could not be distinctly defined and the results of the analysis on the examined dimensions suggest a classification into two fidelity levels.

3.2. Multi-Fidelity

One central idea of the thesis is to develop a process and tool, which allows an easy transition between different levels of fidelity. The initial concept was to enable the forward and backward transitioning between all fidelity levels, which is illustrated in figure 3.8. It was hypothesized that especially the transformation of an artefact to a lower fidelity level could be of interest to systematically steer discussions. For example if a usability walkthrough of a mockup is stuck at the color of a specific button, one could reduce the fidelity and continue the discussion with a wireframe, which would allow to regain the focus on the basic user interaction instead of the design details.

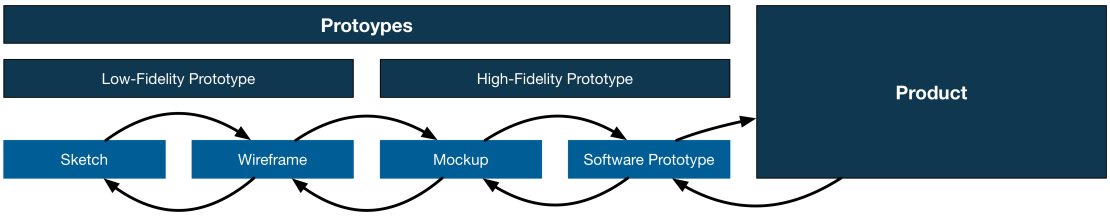


Figure 3.8.: Initial vision for a multif-fidelity prototyping process.

A similar approach was pursued by Coyette in [11], who used four fidelity levels (no-fidelity, low-fidelity, medium-fidelity and high-fidelity) and a tool, which had "a slider [...] [to allow] the user to easily switch between any fidelity level to another" [11]. They focused on the transition between the sketch and the first digital representation of the user interface. Therefore they used a stylus (digital pen input device) and shape recognition algorithm to match the drawn user interface components to a predefined catalog of components. The higher fidelity representations of the component had to be defined explicitly. The export to the final user interface code on the respective platform was enabled through the support of the user interface specification languages UIML (`www.uiml.org`) and UsiXML (`www.usixml.org`). In the course of a semi structured interview the authors analysed the window development time for the different fidelity types. The result had shown, that the times for all four fidelity levels were quite close

with the high-fidelity level being the fastest with an average window development time of 261 seconds. The Computer System Usability Questionnaire (CSUQ) conducted afterwards suggested a moderately appreciated system usefulness and information quality. The additionally collected feedback indicated that the shape-recognition was to slow and a drag-and-drop support of the sketched components was missed. Furthermore most of the participants preferred the high-fidelity level over the no-fidelity level as it makes the impression of being a draft [11].

These results as well as feedback received during the initial thesis presentation at the chair laid to a reevaluation of the previously presented idea for a multi-fidelity prototyping process. The expert interviews presented in section 2.2 also indicated that the current practice is rather to iterate on one specific fidelity level and then transition to a second fidelity level or immediately to the product. This process rarely involves two fidelity levels, most of the time after incorporating the feedback the results are directly passed to the product development. Figure 3.9 illustrates the refined multi-fidelity prototyping process, which more closely resembles the real practice. It should be noted that the transitive relations between the artefacts are possible as well, e.g. immediately continue on the product level after creating a sketched prototype.

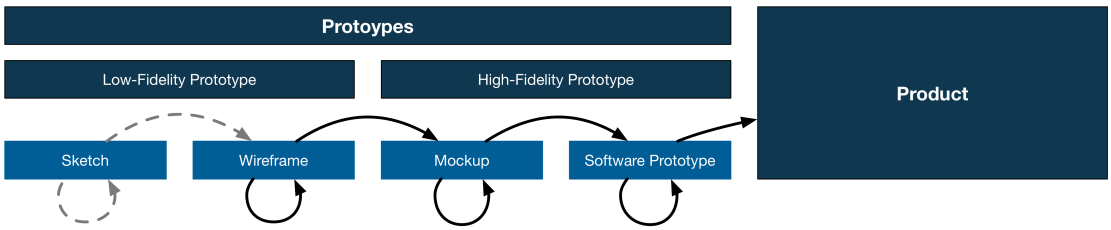


Figure 3.9.: Vision for a multi-fidelity prototyping process.

The dashed and grey coloured transition was extensively analysed in Coyette's paper and was not examined in detail in the course of this research. Nevertheless a short discussion of the results is considered appropriate at this point. The systematic reuse of prototyping artefacts is a key idea of this thesis and thus the approach of recognising UI components drawn with a stylus to transition to the next fidelity-level is well appreciated. However, the analysis of the fidelity-levels in section 3.1 has shown that there is not only a separation into two fidelity groups, but also a separation in the medium used to create them. Typically low-fidelity prototypes are paper-based and high-fidelity prototypes computer-based. Coyette introduced a computer-based technique for the creation of low-fidelity prototypes, which did not yield a positive user acceptance. Interestingly the author himself provided a possible explanation in the course of the related work. User interfaces designed on paper tend to iterate more often and thus create more solution proposals, whereas the prototypes created with a tool on the computer typically tend to work out one solution it its very detail [61]. Fur-

thermore "[low-fidelity] prototyping [...] encourage[s] the stakeholders to focus on the UI interaction rather than on details irrelevant at this level which do not influence the usability" [11]. Low-fidelity prototyping also supports the expectation management as the stakeholders clearly recognise the user interface as not being the final one [35]. This was also one feedback received during the expert interviews. When developing high-fidelity prototypes the polished UI might provoke the expectation that the final user interface is almost finished (see appendix A). The medium paper might support the last two points by communicating a certain non-binding nature. This research argues that there are strong reasons to not "digitise the low-fidelity prototyping" and rather develop a technology, which allows the transformation from paper-based sketches to digital wireframes. With todays visual recognition capabilities and machine learning algorithms this might has become achievable. However, this approach was excluded from the scope of the thesis, but the idea is going to be continued in the course of the outlook (see chapter 8).

In conclusion the multi-fidelity prototyping process is defined as the systematic approach of increasing the fidelity of prototyping artefacts. Typically iterations take place on one fidelity-level and after surpassing a certain maturity level continue to a higher fidelity-level. In practice this often starts with a low-fidelity prototype and continues with high-fidelity prototype till passing over to the product development. Two case studies performed with the industry partner will illustrate this process in section 4.

3.3. Mockup-Driven Development

The transformation of prototyping artefacts to the final user interface code is the last step in the previously introduced multi-fidelity prototyping process and was also specified as an requirement by the industry partner (see section 2.1). During the initial literature research two core papers introducing a "Mockup-Driven Development" approach were identified. The following two sections present the two approaches and subsequently chapter 4 is going to introduce the prototyping process, which combines the multi-fidelity prototyping process with the mockup-driven development approach.

3.3.1. Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering

The paper is motivated by the idea of integrating the benefits of agile software development with the Model-Driven Web Engineering (MDWE) process. Their proposal is to use UI prototypes as a starting point for the modelling process as MDWE "tend[s] to leave User Interface aspects to the end of the development cycle" [47]. Mockups were identified as a key factor in driving the efficiency of agile software development. However, "instead of discarding mockups, [they] transform them into platform-independent UI specifications" [46].

The mockup-driven development process, which the authors named MockupDD process, starts with the creation of a mockup according to specified requirements (see figure 3.10). Afterwards a *Structural User Interface (SUI)* model is derived from the mockup. This is performed in step 2 and supported by the *Mockup Processing Engine (MPE)*, which detects widgets (sets of logically grouped UI elements), determines the hierarchical structure of the widgets and finally detects the layout of the widget within its parent widget. This processing could be skipped, if the mockup was already present in a structured form (e.g. HTML). In the next step a mapping between the requirements in the form of user stories and the SUI model is created by tagging the mockups with annotations. Finally the enriched SUI model is utilised to generate a demo version of the web application and the MDWE models, which are later used for further refinement and the generation of the final web application [46].

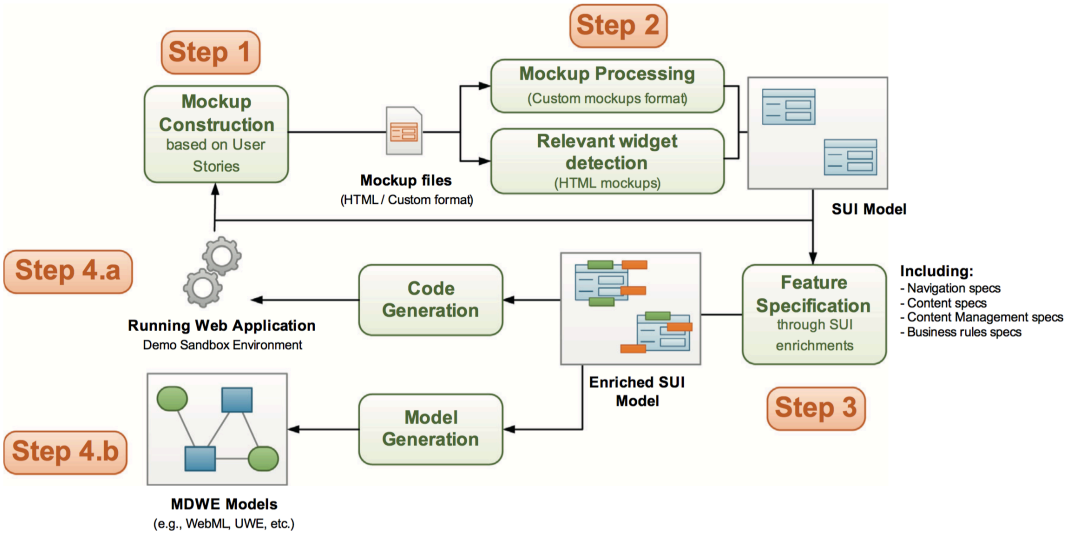


Figure 3.10.: The mockup-driven development (MockupDD) process, including technical steps [46].

This approach of introducing UCD to MDWE solves the issues mentioned at the beginning of the section in an interesting way. Especially the derivation of a structural model from the inherently unstructured mockups is a recurring problem, when enabling a systematic reuse of UCD artefacts, which was elegantly solved. The evaluation has shown that the MockupDD process is easier to learn and more efficient to use then the plain MDWE processes. However, they did not address several issues, which are characteristic for the model-driven approach. Usually the development of user interfaces is not a green field approach and constrained by a given frontend framework like *Twitter Bootstrap* or *Angular Material*. Furthermore development departments might have catalogs of existing widgets to stick to their terminology. The article does

not address this problem. Perhaps the reuse of existing user interface elements could be enabled by the introduction of additional annotations connecting them to elements of the mockup. Model-driven approaches are require a complex tool chain, which has to be adopted to the environment [38]. In this example the tool chain consists of a mockup tool, the MockupDD engine itself and at least one MDWE transformator. The modelling languages are often quite complicated and not suitable for all stakeholders [38]. This is also a valid critique as the authors confessed that their sample users were all experienced MDWE engineers, which might explain the very positive rating on the learnability scale. Nevertheless the MDE approach could provide significant productivity gains, if the knowledge for its establishment is present and potential issues are addressed early on [19].

3.3.2. Mockup Driven Web Development

The second article introduces a declarative approach for mockup-driven development (MDD) with the *Cascading Tree Sheets* (CTS) language. The idea behind this custom language is "the ability to describe the relationship between content and structure on the web" [4] and thus decoupling the content from the structure. The CTS annotated structure is parsed on the client side with JavaScript and enriched with the dynamically loaded content. One aspect of the evaluation is the migration cost from existing CMS systems like *Wordpress* to the mockup-driven system. The results were not available yet, but the prepatory scrapping of Wordpress themes was completed. A usability study indicated that the approach is significantly faster for reuse tasks. Furthermore a performance analysis has shown that it could yield a four-fold throughput improvement for queue-heavy workloads (e.g. blogs) [4].

In summary this research introduces an advanced client-side templating engine. The term mockup-driven development is used in a quite different context then in the previously presented article. In the course of this work a mockup depicts the structure and thus an input for a web application, which has to be combined with its content. The term of fidelity is not mentioned, but considering that the mockup is the template for the real application one could assume that it has to have a extremely high-fidelity, if not even being the real product without its content. Furthermore this approach requires the use of HTML-based mockups as the CTS language annotates the HTML-tags. In comparison to the MockupDD approach this article lacks the consideration of mockup tools, which do not operate on HTML code. Furthermore the MDD technology is not embedded in a process. Nevertheless the decoupling of content and structure is an interesting aspect for a mockup-driven development approach. More advanced prototyping tools like *Justinmind* allow the injection of data into the mockup following a similar approach. However, an export of UI code or the data model is not supported.

3.3.3. Assessment of the Approaches

First of all the *Cascading Tree Sheets* concept has a very different understanding of the term mockup then the one established in this thesis. Instead of an artefact of the UCD process a mockup is rather treated as an abstraction of a user interface, which is lacking the content. However, this separation enables the reuse of existing UI elements, which was a prominent requirement for the solution design. In the course of the following chapter this idea is going to be considered as part of the definition of reusable components.

With regard to the multi-fidelity approach both articles are focusing on the transition of a high-fidelity mockup to the final product. This step is only one part of the whole UCD process to be developed and did not receive the highest priority during the requirement elicitation. The *MockupDD* process is embedded into the agile software development, but the CTS are rather described as a technology then a part of the UCD process. This thesis is going to consider a multi-fidelity approach and not only the final transition to UI code. Furthermore its focusing on the collaboration between all involved stakeholder and incorporate the surrounding environment.

4. Prototyping Process

Under the consideration of the management's requirements (see section 2.1) and the results of the expert interviews (see section 2.2) as well as the related work about mockup-driven development a custom prototyping process was developed. Figure 4.1 gives an overview over the participating systems and roles. In section 4.1 the process is explained in detail.

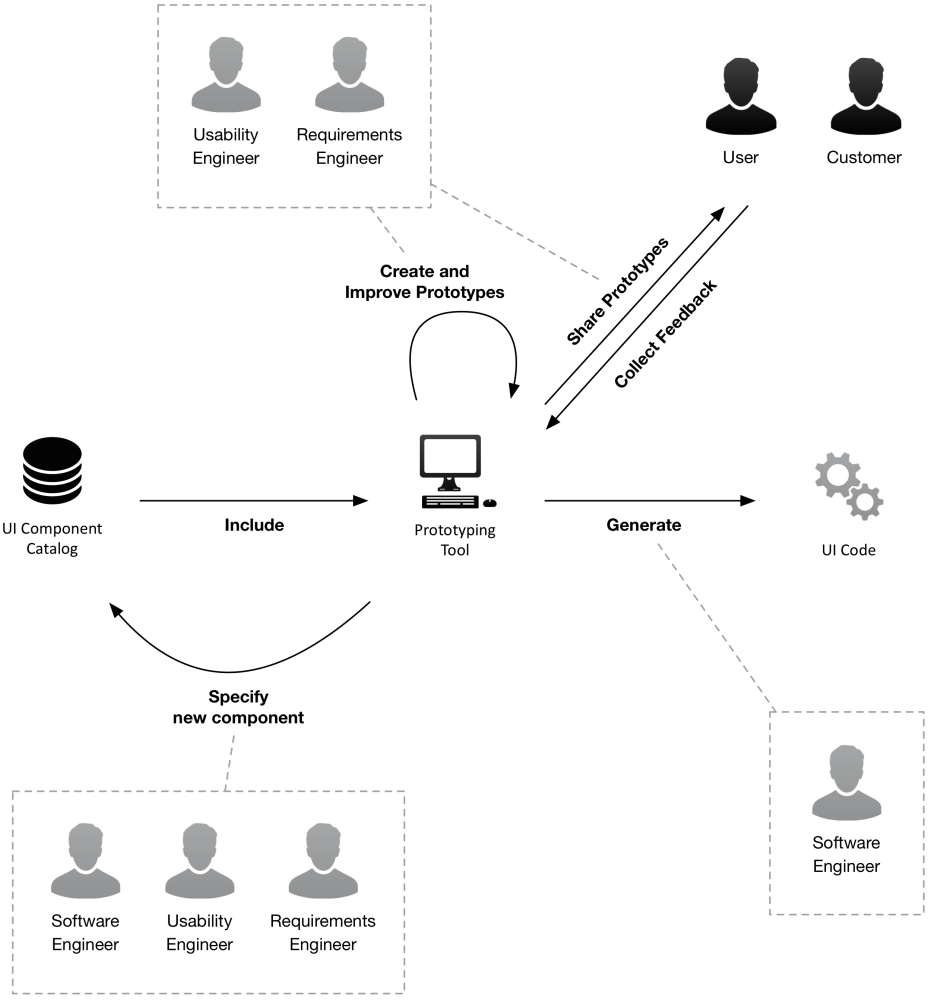


Figure 4.1.: Overview of the prototyping process the participating systems and roles.

As emphasised before the mockup-driven development approach is too narrow to solve the identified problem. Thus a custom prototyping process was defined and according the term definitions introduced before is called *prototype-driven development*. The process is independent of the utilised fidelity-level and could, with some constraints, be executed with any prototyping tool (sketch, wireframe, mockup, etc.). In figure 4.1 the participating roles are marked by grey dashed lines and associated with the process steps, which they are involved in.

The requirements or usability engineer identifies a requirement which involves the creation of an user interface. This process does not make any assumptions of how these requirements are organised to allow the independence of the applied software engineering process. Depending of the structure of the requirements and project the requirements engineer creates a new prototype for the feature or adds an additional view to an existing one. The prototyping tool used for this purpose supports the engineer by providing a catalog of existing UI components. It is now the task of the requirements engineer to identify and layout the components, which are necessary to implement the requirement. If the component catalog does not suffice, a new component could be specified. At this point the software engineer joins the process to discuss the new component together with the requirements engineer. This collaboration is extremely important to assure, that the new component could be implemented under consideration of the prevalent constraints (technology, security policies, corporate design, etc.). Furthermore the aspect of reusability has to be discussed. Is the component specific for the current project or could it be generalised for different use cases? The requirements engineer adds a placeholder component to the prototype, so that he is not blocked until the new component is available in the catalog. Within this placeholder any form of representation of the new component could be created and he is encouraged to do so, because this reflects the basis for the discussion of the new component with the software engineer. After completing the prototype the requirements engineer shares the draft with the users or customer to collect feedback. Depending on the received comments or performed assessment the prototype is refined or released for the implementation. The refinement step is executed iteratively till a specified maturity level of the prototype is reached (e.g. if an interview was conducted the prototype could be passed to the development department, if less then two comments for improvement were expressed). Obviously this threshold is dependent on the concrete environment and needs to be balanced with a cost-benefit analysis. If too few iterations were performed, changes might not be detected and occur in later stages of the project. If too many iterations are performed one might end up applying the *"design-paralysis"* anti-pattern (analog to the *"analysis-paralysis"* anti-pattern [7]), delay the project and / or exceed the budget. In the final step of the prototyping process the software engineer receives the prototype and is able to transform it to UI code and scaffold the further development.

4.1. Process

After giving an overview in this section the prototype-driven development process is presented in detail. Therefore the process was formalised using the Business Process Management Notation (BPMN) (see figure 4.2). An enlarged version of the process chart could be found chapter C of the appendix.

There are three roles participating in the process. The role of requirements engineer, which could also be taken by an usability engineer, but is omitted for simplification reasons. Furthermore there is the software engineer and the user of the future system. The process starts with the receipt or analysis of a new requirement by the requirements engineer. In the first step he has to decide if the requirement affects the user interface of the system. This means, that he has to identify, if the feature requires the adjustment of an existing UI or the creation of a new one. If the decision is negative, the requirement is not of interest for the prototype-driven development approach and could be handed on to the conventional requirements engineering process (e.g. for prioritisation and assignment). In the positive case the subprocess for the creation of a new prototype is instantiated. The attached chart only considers the creation of a new prototype, but it is trivial to extend the process with a task for the retrieval of the existing prototype. The circular icon on the bottom of the subprocess indicates that it could be executed iteratively. This refers to the dimension of time – the feedback loop to incorporate the comments of the users – as well as to the structural dimension, which means that several iterations for the different components of one view might be necessary. For each of those components the requirements engineer has to check if there is an existent component in the catalog. In the positive case the component is reused and laid out into the view. Otherwise, in the negative case, the subprocess for the specification of a new component is initiated, which is described in section 4.1.1. After finishing the design of the prototype it is shared with the users of the system. The user reviews the prototype and provides feedback to the requirements engineer, who after reviewing decides if he has to improve the prototype or if it is stable and ready for the handover to software engineer. Finally the developer utilises the prototype to generate the code of the user interface and continues with the conventional development process.

The description of the process was intentionally formulated in an abstract way to be independent of a concrete implementation with certain software tools or UCD methods. In section 4.2 two case studies conducted during projects of the industry partner will be presented. These case studies serve two purposes. Firstly they are used to discuss, if the suggested process could be actually implemented in an enterprise environment. Secondly the case studies, although not yet complying with the proposed process, partially illustrate how specific steps of the process could look like in reality.

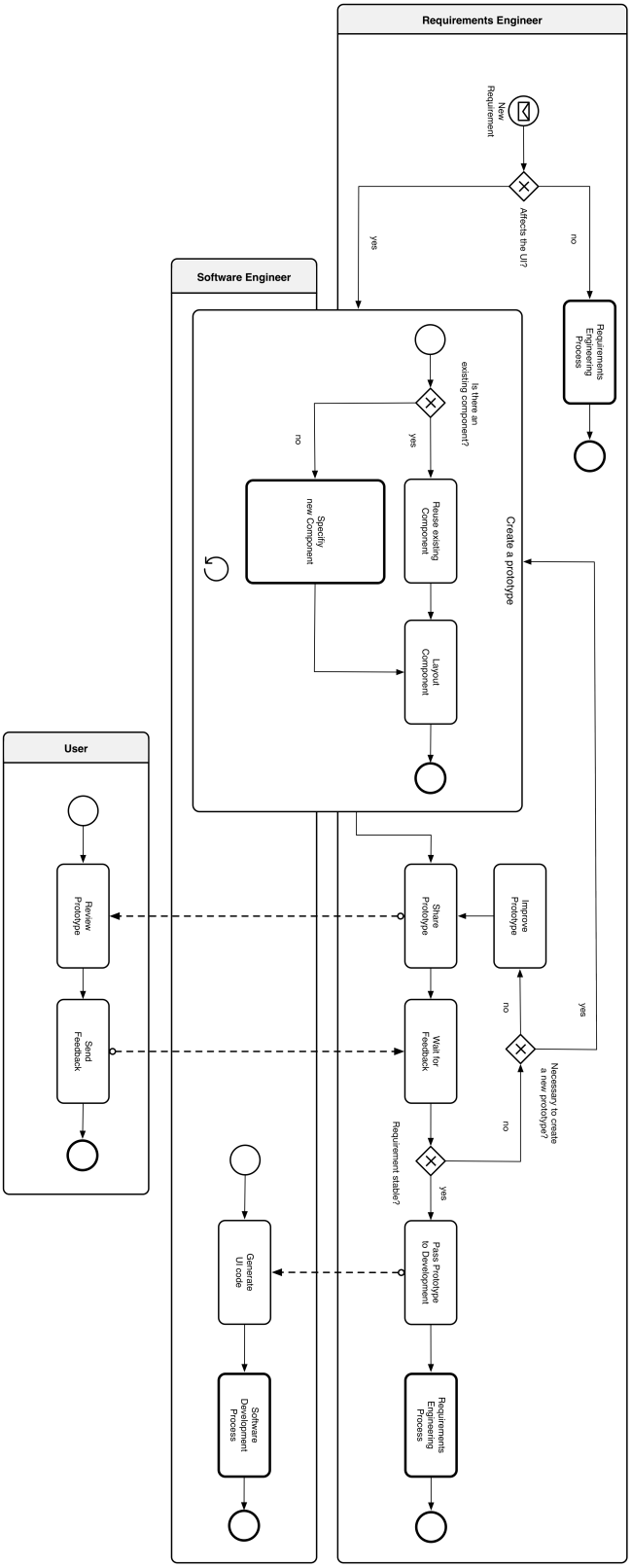


Figure 4.2.: The prototype-driven development process.

4.1.1. Component Specification Subprocess

Before continuing with the case studies the subprocess for the specification and creation of a new component should be shortly introduced (see figure 4.3). Just as with the previous chart, an enlarged version could be found in the appendix (see chapter C).

The subprocess involves the same roles as the parent process and has a similar structure. Instead of the single feedback loop with the user an additional one with the software engineer is upstream to analyse the technical feasibility of the specified component. After passing this quality gate the usability of the component is analysed with the user. In both cases a negative result leads to an improvement of the component with the aid of the received feedback. In the case of a positive usability the component is passed on to the software engineer. Analog to the parent process the developer uses the prototype to generate the user interface. Afterwards he implements and connects the business logic to the UI. Finally the software engineer publishes the component to the shared component catalog. It should be noted that the specification of the component's business logic is not covered within this approach and needs to be addressed by the conventional development process. Furthermore an additional verification of the component's usability after the completion is imaginable and reasonable, but for the sake of simplicity not considered in the subprocess chart.

4.2. Prototyping Case Studies

This section introduces two case studies of projects which relied on UCD, but were not yet implementing the suggested prototyping process. Each case already considered partial steps of process and serve as an example, but the main goal of this presentation is to analyse, if the process could be theoretically implemented in a real world scenario. During both projects prototypes of a varying degree of fidelity were created and informally discussed with the stakeholders. The measures were executed by the author of the thesis in the role of a usability engineer. Time-wise the UCD activities were conducted at the beginning of the project and in the case of the mobile application stretched far into the development phase. This was possible due to the agile development approach, which facilitated the consideration and implementation of user feedback during all stages of the project.

4.2.1. Case Study 1: SIPCA - Web Application

SIPCA is an application, which processes the variable compensation of employees according to their personal goals. This tool should be extended to incorporate the target setting for whole countries and departments as well as the management of special effects, which prevent or affect the achievement of the targets. The demand was based on an existing software solution, thus the functional requirements were quite stable.

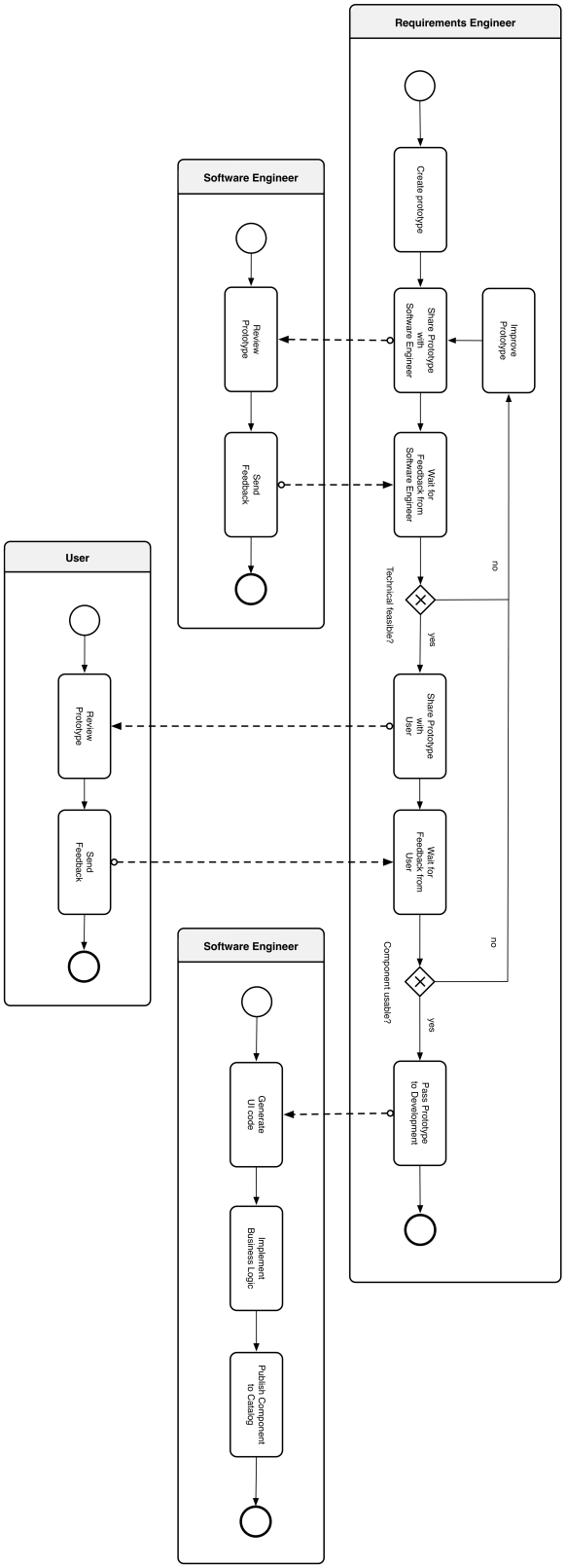


Figure 4.3.: The subprocess for the specification of a new component.

However, the user interface of the application should be redesigned to follow the corporate design guidelines and address several usability issues, which occurred in the existing solution. Thus the rational behind the creation of high-fidelity mockups was primarily to communicate the new application design and in the course of this improve the usability the existing solution.

The requirements were collected and documented in the form of a slide deck during a workshop with the customer and a requirements engineer. Afterwards the requirements were preprocessed by a second requirements engineer and a first draft was entered into the department's ALM solution – *HP Quality Center*. The application was split into two bigger modules and several feedback meetings between the usability and requirements engineer took place to clarify open issues. First paper-based sketches were discussed and afterwards recreated as high-fidelity mockups with the prototyping tool *Justinmind* (see figure 4.4 and 4.5). After finishing the mockups for the first module a meeting with the responsible software developer was scheduled to discuss the technical feasibility of the created designs. There were only minor adjustments necessary as the author of the thesis also has the role of a software engineer and thus is familiar with the available UI components. Following this, a walkthrough with one user and two customers was performed. The walkthrough was realised with the internal live conferencing solution and not in person. However, during the session several improvements were identified and incorporated into the second version of the mockups. Furthermore the requirements were updated accordingly by the requirements engineer, who took part in the walkthrough as well. Finally the mockups for the second module were completed and the just described process was repeated with the small distinction, that the previously discussed adjustment were shortly presented at the beginning of the second walkthrough.

The usability approach was viewed very positively by all participants and the requirements engineer made the comment that especially the early feedback from a second person with a different perspective helped to identify ambiguous and unclear requirements. In this case the high-fidelity mockups did not create the expectation of an almost finished product. The UI of the final solution only differed slightly from the mockups and no major changes in the requirements ocurred during the development.

If one compares the presented case with the proposed process, one could see that the feedback loop with the user was established and executed in the form of a usability walkthrough. Furthermore the collaboration between the requirements engineers and the software development department was successfully established through the usability engineer. Two coordination meetings ensured that the imagined solution could be implemented with the given constraints. However, the creation of the mockups and also the previously mentioned coordination could have been faster, if the UI components were already present in a shared catalog as suggested. The utilised tool *Justinmind*

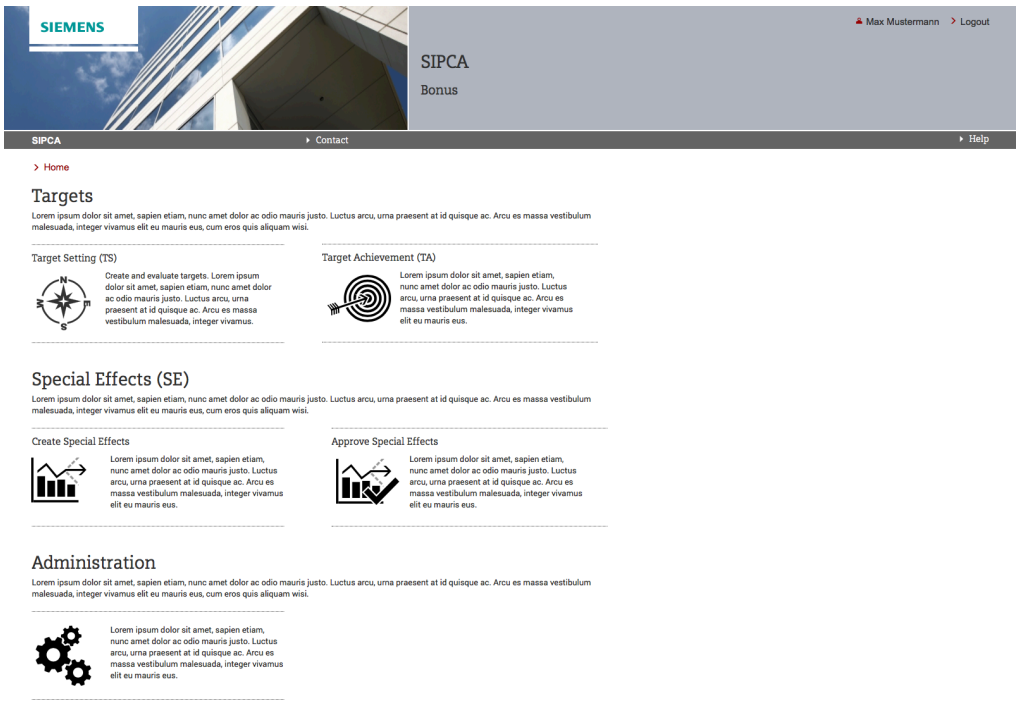


Figure 4.4.: High-Fidelity mockup of SIPCA's landing page.

supports the creation of a custom component catalog, but it is has to be maintained manually.

4.2.2. Case Study 2: Siemens Corporate Directory - Mobile Application

The second use case was the creation of a new concept and implementation for a mobile app to access the contact information of all Siemens employees. There was an existing solution, which only supported the iOS-platform. Thus one major requirement was to additionally support Android and improve the usability of the existing solution (see figure 4.6). The critique of the existing app was that it does not comply with the corporate design, the functional scope was very narrow and the detail screen of the employee was crowded with unnecessary information. Furthermore the search failed to find correct entries in many constellations, which were successful on the corresponding web application.

For this scenario paper-based sketches were created to collect ideas and feedback. Besides that the internal social network was scanned to collect user feedback and improvement suggestions. The concept proposed to include the employee's image in the detail page and to reorganise the contact details by starting with the commonly used ones. Furthermore the local time of the employees home location as well as the complete address should be listed. The sketches were the basis for a first presentation of the

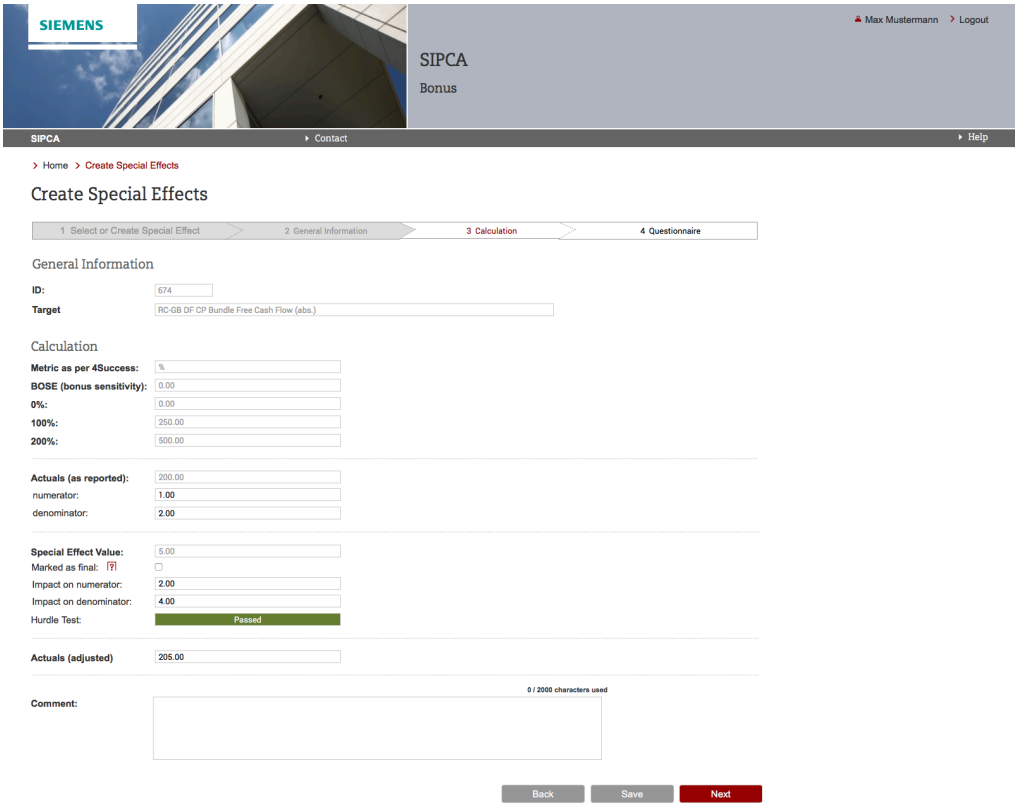


Figure 4.5.: High-Fidelity mockup of SIPCA's special effect creation process.

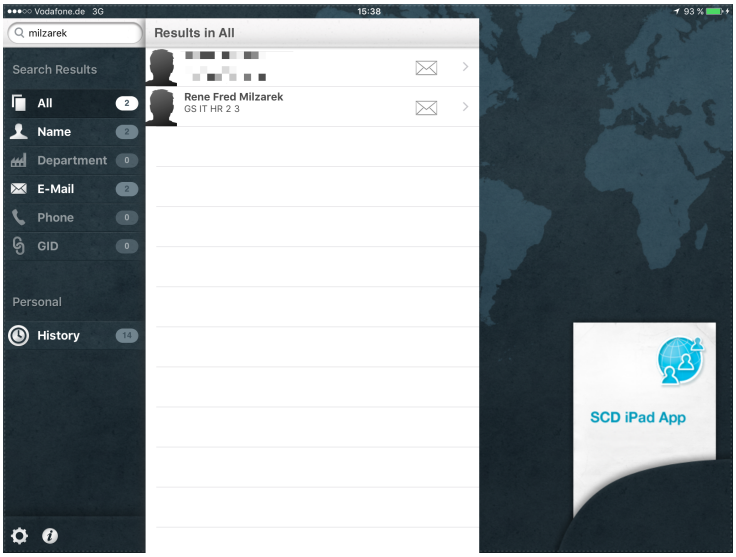


Figure 4.6.: Screenshot of the old Siemens Corporate Director app on an iPad.

improvement potentials at a meeting with the head of the department responsible for the service. On the technological side the decision was made to use a new hybrid app framework to be able to address the cross-platform requirement. This also accounted for the decision to evaluate the technology in the course of a software prototype, which was comprehensively implemented by two interns.

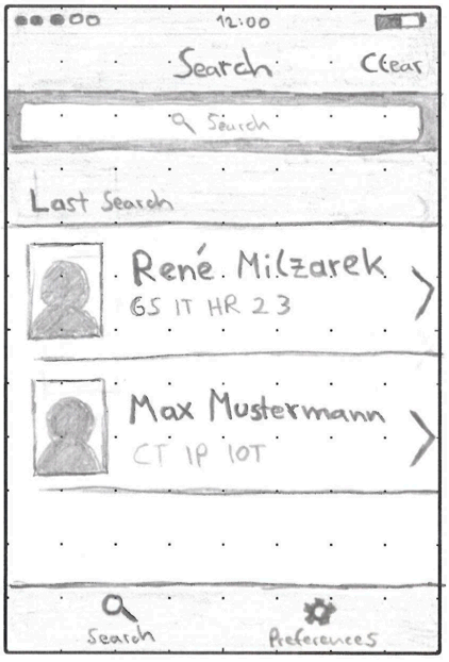


Figure 4.7.: Sketch of the search result view.

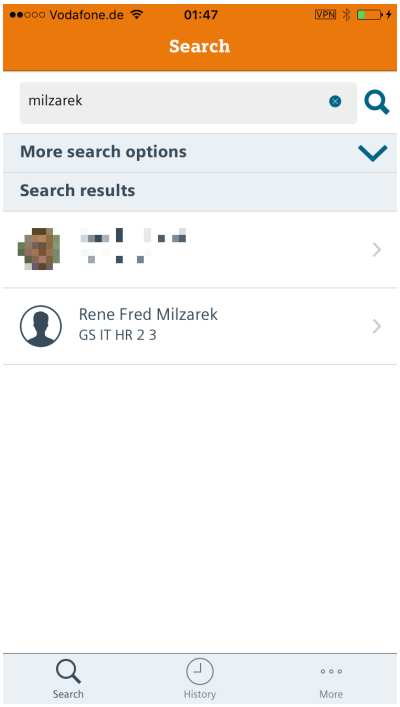


Figure 4.8.: Search result view implementation.

The prototype created the impression of being a production ready solution, but the impression was misleading. After the handover of the app to two software engineers a major instability was detected, which caused random crashes of the whole application. The cause of the defect could not be identified precisely and did not leave any stack traces. The code imported several unused libraries and the overall code quality made the error detection difficult. Finally the solution was a rewrite of the app, which resolved the instability. Despite these issues the new app was able to implement the suggested improvements and the users were very satisfied.

Regarding the prototyping process this case had a transition from a low-fidelity sketch to a high-fidelity software prototype and did not systematically assess the usability with users of the app. However, several improvement suggestions were indirectly collected from the user through the internal social network. In contrast to the first example the project had a explorational character on the dimension of the functional

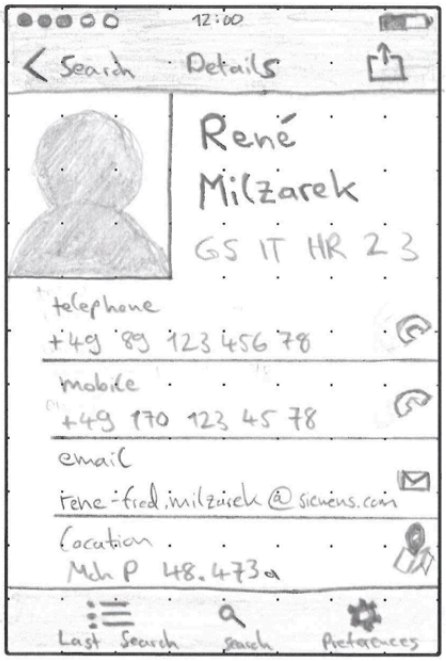


Figure 4.9.: Sketch of the employee's detail view.

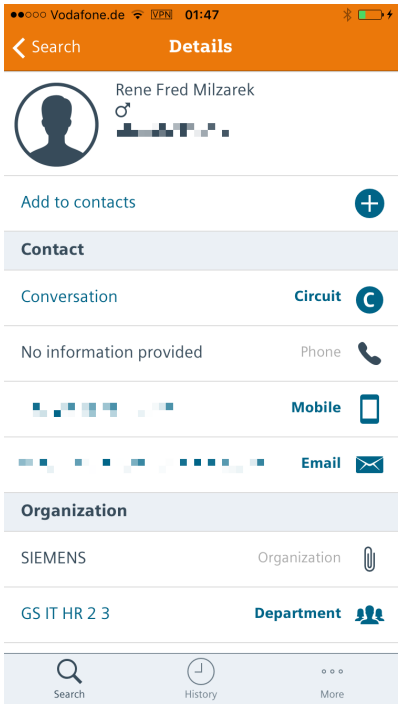


Figure 4.10.: Detail view implementation.

scope as well as the dimension of the technology. A software prototype is the perfect mean to evaluate these aspects. However, the conduction of a standardised usability questionnaire with the old and new solution would have provided the chance to exactly measure, if the assumed usability improvements were created.

4.2.3. Evaluation of the Case Studies

Both cases illustrated that certain steps of the process are individually implemented, but there is no coherent integration of the steps. The prototyping tool *Justinmind* has a component catalog, which is not synchronised with the actually available UI components. In the first example the mockups are unfortunately only used as a requirements document, although they are based on existing components. The missing association between the prototypes and the real components complicates a systematic reuse of the artefact.

5. Implementation

The case studies also supported the prioritisation of the requirements elicitation and attributed a high priority to the aspect of collaboration and the shared component catalog. Following the suggested prototype-driven development approach, the systematic creation and maintenance of such a catalog is a key prerequisite to enable the reuse of prototypes for the UI generation. Therefore, the UI component catalog was further analysed in the course of a prototypical implementation. The following requirements were deducted from the prototyping process and, if applicable, inherited from the requirements identified in chapter 2.

- Create a component catalog
- Specify new components
- Enable the reuse of existing components
- Allow collaboration
- Support multiple fidelity levels

5.1. Component / View Model

An important aspect of the implementation was to identify and implement a model for the components / views. Jonathan Allen defines the view model "[..] as a surrogate data context" [2]. The real data is passed to the view via one or multiple properties.

Initially, a user interface modelling language had to be identified for the application or adaption. Over time a variety of user interface modelling languages have been developed: Maria [43], UML (`http://www.uml.org/`), UMLi [45], UsiXML [29] (`http://www.usixml.org/`), DiaMODL (`http://www.idi.ntnu.no/~hal/research/diamodl`), Himalia [60], UIML [1] (https://www.oasis-open.org/committees/uiml/). The choice of the suitable modelling language itself is an individual research topic [39]. The decision on a view model was not only relevant for this thesis, but also for two more, which were supervised by the same advisor. During a joint workshop the User Interface Modeling Language (UIML) was defined as a basis for the view model and slightly adapted. The UIML provides a structure, which excellently fits the problem. An interface element consists of a structure, a style, a content and a behaviour (see figure 5.1). Thus,

one component is defined by its structure, data or content properties, style properties and behaviour definitions.

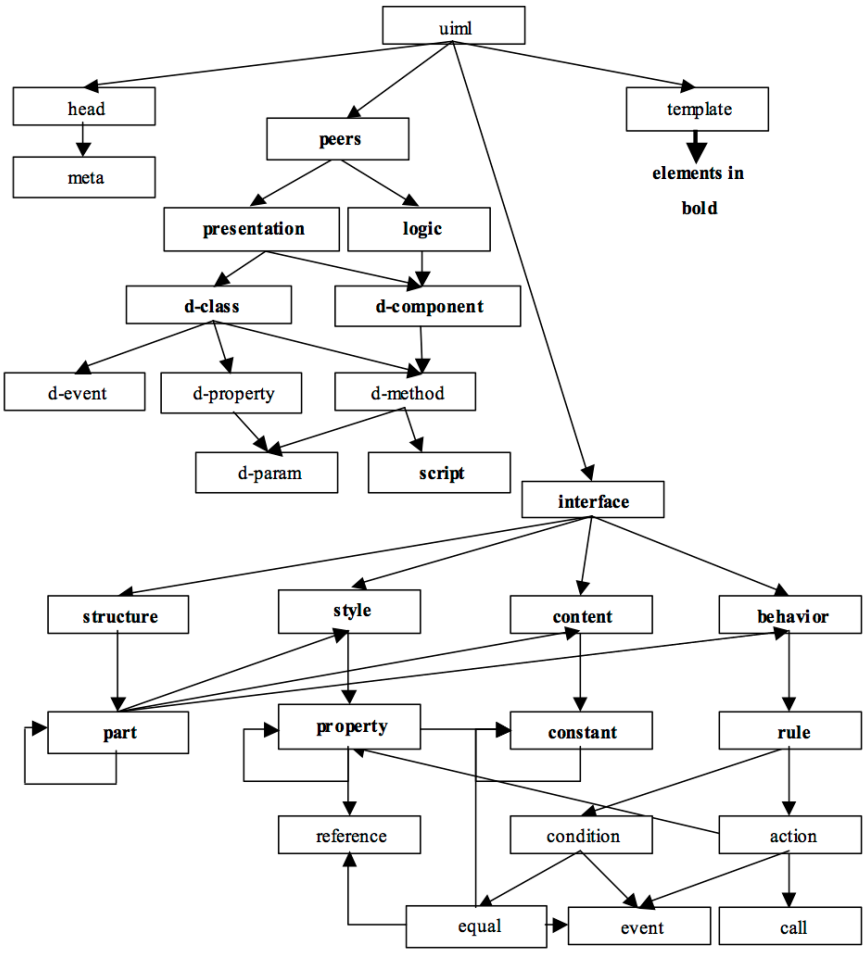


Figure 5.1.: The elements of the UIML2_0e DTD [44].

The relevant elements for the definition of a component were extracted from the UIML and are illustrated in 5.2. Within this thesis the behaviour of a component was omitted and the focus was set on the style properties. The relationship between a property and the abstract part, which could be a reference to another component, allows the overwriting of the style properties of an instantiated component. The properties were not constrained in any way, e.g. the names have not to be chosen from the set of valid Stylesheet attributes. In the concrete implementation the structure was stored in the form of a JSON object, which referenced other component instances by their identifier. By default a new component has a structure consisting of exactly one element, which is an instance of the component itself.

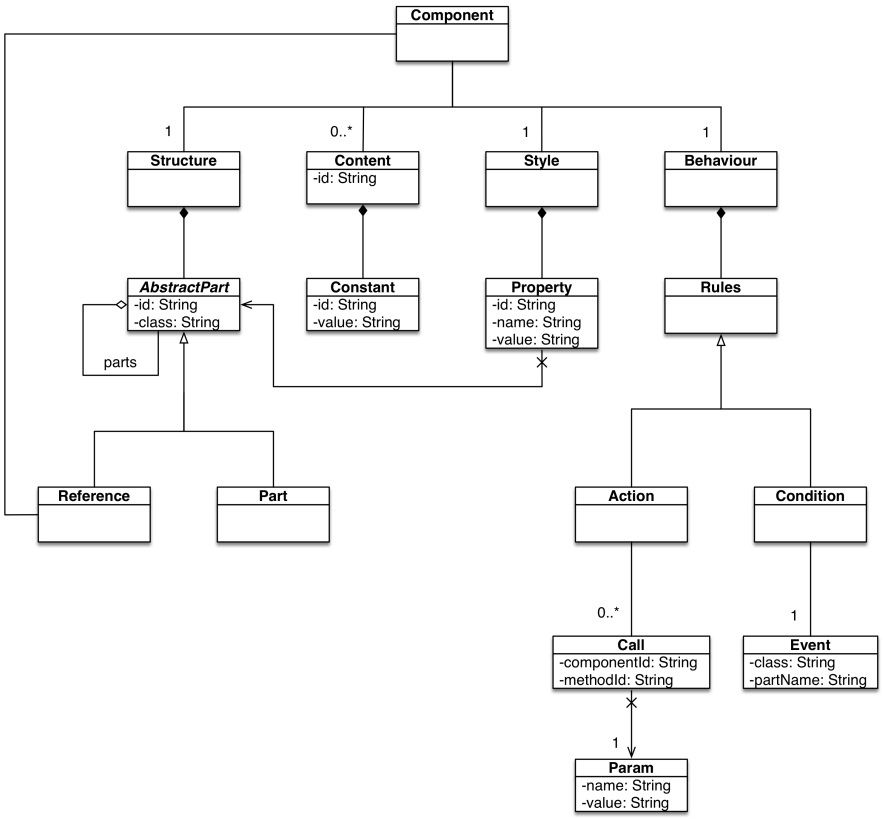


Figure 5.2.: Class diagram for the specification of components.

5.2. System Architecture

The component modeller – called *Proteon* – relied on Parse (`https://parse.com/`), which is a *"backend as a service"* solution, to quickly and easily define a REST API. The backend was deployed in the form of a *Docker* container composition. Internally *Parse* communicates with a *Mongo* database to persistently store the classes. As containers do not have a persistent storage, Mongo's data was stored on an external data volume mount. Furthermore, *Parse* provides a web frontend – the *Parse Dashboard* – to comfortably create classes and maintain objects. Through this web application the classes were iteratively refined and the so defined REST API was consumed by an *AngularJS* frontend.

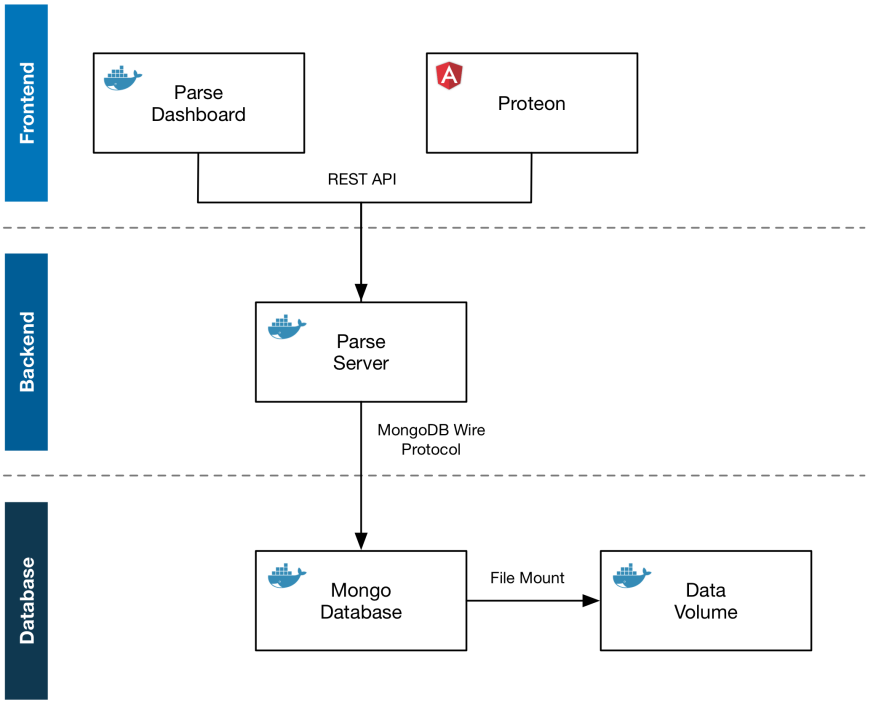


Figure 5.3.: System architecture of *Proteon*.

5.3. Technical Implementation

The user interface of the web application was designed with paper-based sketches, which were then discussed with the advisor (see figure 5.4). The development followed an agile approach with multiple iterations. A first version of the user interface was created on basis of the sketches. In figure 5.5 the detail view of a component is shown, which provides the option to drag and drop new child components from the list on the

lefthand side to the component structure tree on the right side.

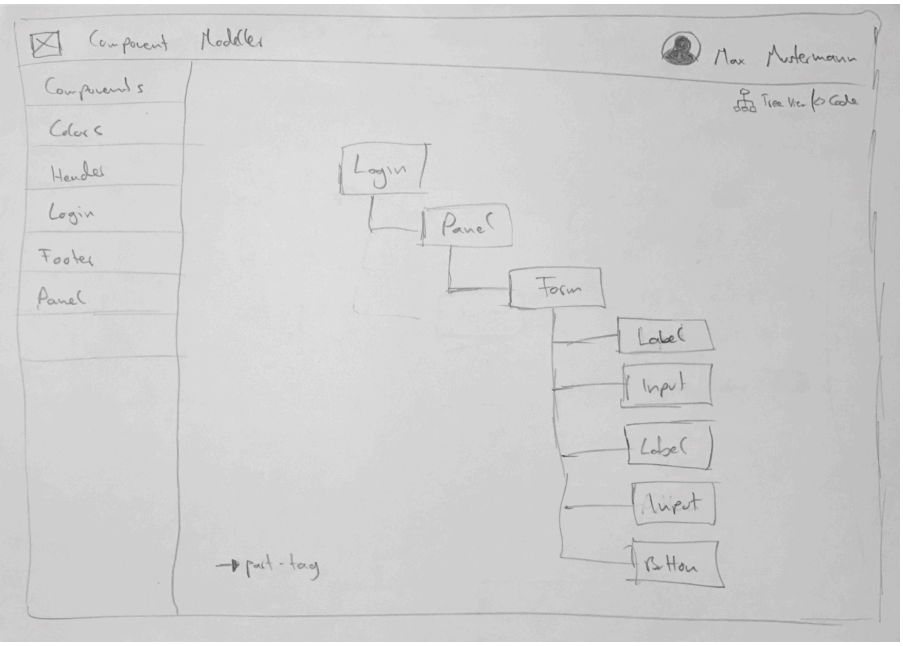


Figure 5.4.: Paper-based sketch of *Proteon*'s component's structure tree.

During the the presentation of this screen the feedback was collected, that the component structure tree, as the core element of this view, should receive more space in the UI. The decision was made to improve the layout by replacing the drag and drop interaction with a submenu. A button at each component of the structure tree allows the access to the submenu. Within this menu the user could search for components and through a click on the component add a new child component to the respective element of the structure tree (see figure 5.6). This redesign allowed the tree to occupy the whole width of the central content area of the application.

Finally the tool was presented in the context of the prototyping process and the usability was measured with the System Usability Scale (SUS) questionnaire. The results of the evaluation is going to be introduced in chapter 6.

5.3.1. Third Party Libraries / Software

As previously mentioned, the software prototype relied on several third party libraries. This section gives an overview of the included libraries and software tools. The backend as a service – *Parse Server* – is licensed under the BSD license. The *Parse Dashboard* has an individual license agreement, which allows the non-exclusive royalty-free use, reproduction, distribution and modification for internal use. The *Docker Compose* setup was repurposed from the *Github* user "yongjhih", who licensed the scripts and

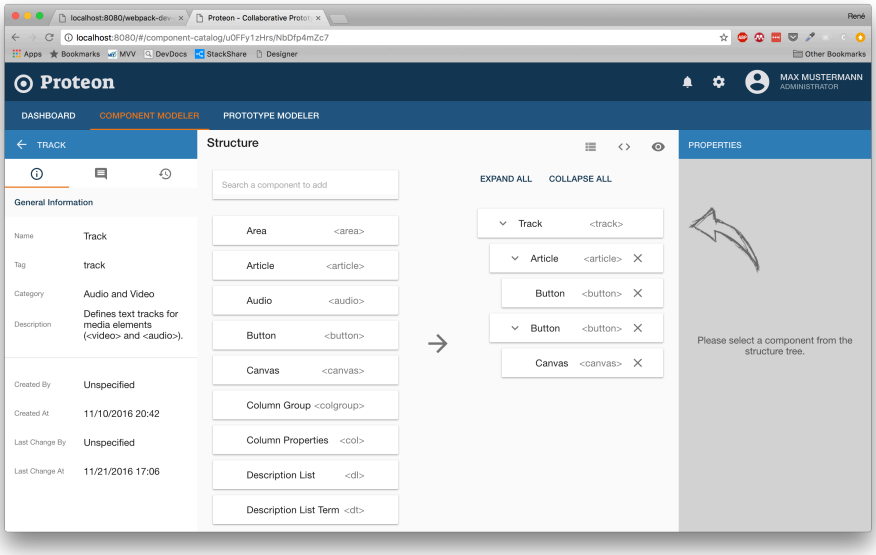


Figure 5.5.: Implementation of the component's structure tree with a drag and drop interaction.

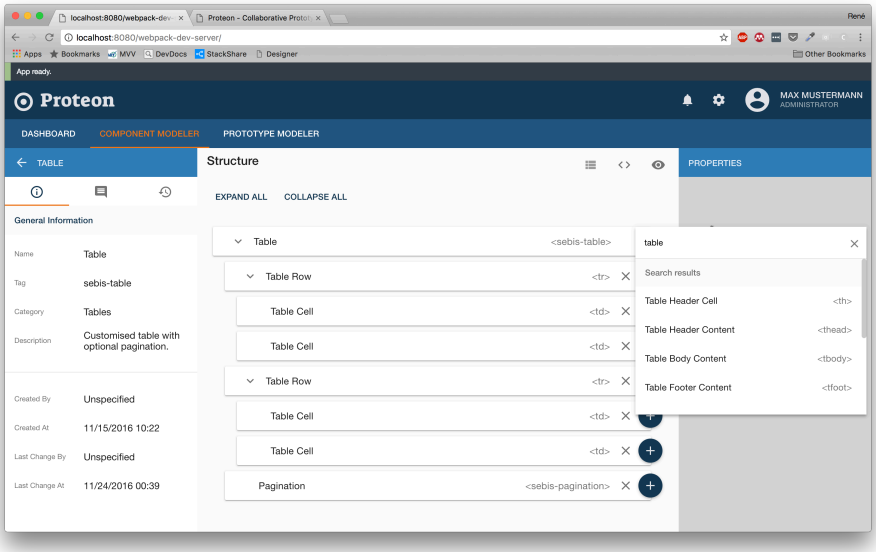


Figure 5.6.: Implementation of the component's structure tree with the interaction through a submenu.

configurations under the Apache license, version 2.0 (`https://github.com/yongjhih/docker-parse-server`). Furthermore the frontend relied on 9 different libraries, which were with the exception of one all licensed under the MIT license. The complete list could be found in the appendix D.

5.3.2. Execution Instructions

The project was managed in a public Git repository on *BitBucket* (`https://bitbucket.org/neotreat/masters-thesis-code`) and is licensed under the MIT license to allow further research and the continuation of the development and integration at the industry partner. As mentioned before the web application's backend is based on a setup of multiple *Docker* containers. The relationship between the containers is configured in a *Docker Compose* file and the environment variables were extracted to an environment file (see listing 5.1).

```
1  ##
2  # @author René Milzarek <rene.milzarek@in.tum.de>
3  # @copyright Copyright 2016 by René Milzarek.
4  ##
5
6  APP_ID=<<APP_ID>>
7  MASTER_KEY=<<MASTER_KEY>>
8
9  PARSE_DASHBOARD_ALLOW_INSECURE_HTTP=1
10 SERVER_URL=http://localhost:1337/parse
11 USER1=<<USER>>
12 USER1_PASSWORD=<<PASSWORD>>
```

Listing 5.1: Environment configuration in `backend/.env-default`.

Create a copy of the environment file in `backend/.env` and provide the missing parameters. It is recommended to use the application identifier "Proteon". The master key should be chosen randomly and only shared with trusted parties. The user credentials will be used to access the *Parse Dashboard*. Afterwards the backend could be started with the following commands.

```
1  cd ./backend
2  docker-compose up
```

Listing 5.2: Startup commands for the backend.

The same configuration has to be provided to the frontend. For this purpose open the `frontend/src/app.js` file and go to line 44 where the code snippet of listing 5.3 is located. Copy the master key to the marked location and save the change.

```
1  // ...
2
3  // Setting global constants
4  appModule.constant('CONFIG', {
5      'APP_ID': 'proteon',
6      'API_KEY': '<<MASTER_KEY>>', // = JavaScript key
7      'API_URL': 'http://localhost:1337/parse'
8  });
9
10 // ...
```

Listing 5.3: Environment configuration in `frontend/src/app.js`.

Before starting the web application the third party libraries need to be installed by the execution the following commands (see listing 5.4). This has to be done only once.

```
1 cd ./frontend-web-app
2 npm install
```

Listing 5.4: Installation of the frontend's third party libraries.

Finally, the frontend could be started by running the `webpack-dev-server`. After several seconds the web application is going to be available at `http://localhost:8080`. Please note, that the *Parse Dashboard* is going to reachable at `http://localhost:4040` and requires the previously defined credentials for the first access.

Part III.

Evaluation

6. Evaluation

The prototype-driven development process and the software prototype *Proteon* were evaluated in the course of a usability walkthrough and a subsequent conduction of the System Usability Scale (SUS) questionnaire. At the beginning of the walkthrough the prototyping process was explained on the basis of the process overview diagram (see figure 4.1). Special attention was paid to the description of the participating roles so that the interviewee was able to identify where he or she is interacting with or executing certain steps of the process. It was explained, that the whole process should be evaluated, but one aspect – the component modeller – was additionally implemented in the form of a software prototype. Afterwards the modeller was presented to the interviewee and the following aspects were explored with the user. The dashboard of the web application was always presented as the first screen. Afterwards the overview of component catalogs was opened and it was explained that there are several predefined component catalogs (e.g. HTML5 elements), which could be used as building blocks for new components. In the next step the company specific component catalog was accessed and the details of the table component viewed. The comment functionality on the component level was demonstrated afterwards and finally the user explored the creation of a new component and properties on various fidelity-levels.

The walkthrough was concluded by an online questionnaire, which consisted of the standard usability questionnaire SUS and additional 7 items to analyse, if the suggested process solves the identified problems. The decision for the SUS questionnaire was made for its simplicity and thus ability to be performed quickly. The questionnaire consists of 10 items and could be conducted in a short period of time. This aspect was important to assure that the complete walkthrough does not exceed 30 minutes. The custom questions intentionally followed the structure of the questionnaire in the form of "I think that..." questions to not irritate the interviewee. Furthermore the same 5 point Likert scale was used (1 indicating strong disagreement and 5 strong agreement) for the answer options. Also the evaluation scheme of the SUS questionnaire was considered, which causes the fact, that the values only range from 0-4 in the following sections. Please refer to [6] for a complete explanation of the applied evaluation method.

The detailed analysis of the collected answers and feedback could be found in the appendix E. At this point the most relevant results should be discussed. Overall $N = 8$ persons participated in the usability walkthrough. They were all employees of Siemens and represented 4 different roles (software engineer, requirements engineer, team lead

and operations manager). The completion rate of the interviews and subsequent questionnaire was 100%. There occurred no issues during the elicitation, which would have invalidated a data entry. The following evaluation is structured according to the online questionnaire in the three sections suitability, usability and feedback. The first section encompasses the custom questions, which were designed to examine the fulfilment of the identified requirement. The second section covers the SUS questionnaire and the final one presents the feedback, that was additionally provided by the participants.

6.1. Suitability

An improvement of the collaboration between the department and participating roles of the software development process was the requirement with the highest priority. Therefore the answers to the question, if the collaboration could be increased with the suggested prototyping process and the component modeller should be presented first. Overall 6 (75%) of the persons strongly agreed, that the collaboration could be improved by the suggested solution (see figure 6.1). The mean value was 3.5 ($\sigma = 1.07$) and supports the hypothesis that the collaboration could be improved.

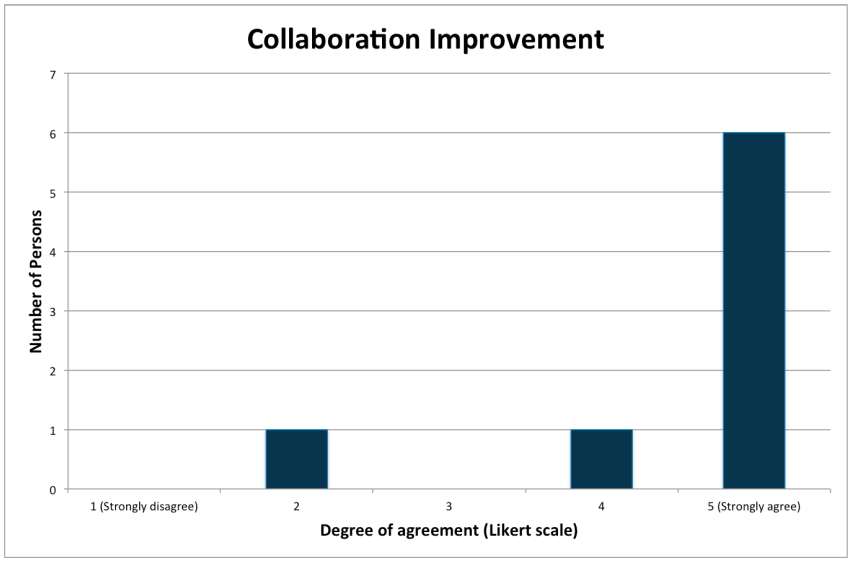


Figure 6.1.: Evaluation of the collaboration improvement.

The second most important requirement was the establishment of a shared component catalog. During the interviews in the course of the problem identification the increased collaboration and a more systematic reuse of existing components was associated with this requirement. Thus the question asked targeted the enhancement of component reuse. Here the interviewee had the exact same impression and overall

75% (6 persons) strongly agreed to the statement, that the component reuse could be improved (see figure 6.2). Accordingly the mean had the same value of 3.5 ($\sigma = 1.07$).

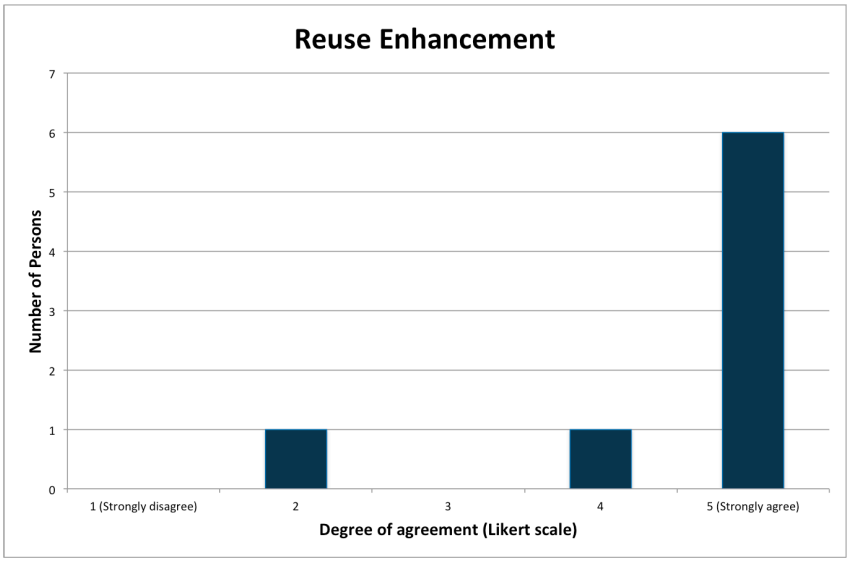


Figure 6.2.: Evaluation of the reuse enhancement.

The next questions asked the participants to evaluate, if the prototyping process could accelerate the software development. This questions is related to the previous one, but had the intention to verify, that the suggested solution is not too complex and slowing down the stakeholders. The majority of 4 persons (50%) agreed with the statement, that the software development could be accelerated (see figure). The mean was with a value of 3 ($\sigma = 0.76$) lower, but nevertheless indicates an average agreement. However, the distribution of the answers was slightly smaller.

Afterwards the error-proneness was examined. It should be noted that this statement was negatively formulated: "I think that the system is prone to errors", which requires the given answers to be evaluated inversely with the formula $5 - x$. For more details on the evaluation method, please refer to [6]. Overall 4 persons (50%) disagreed with this statement (see figure 6.4). The mean had a value of 2.5 ($\sigma = 1.07$), which is considerably closer to an average rating.

The question about the ability of the process to be integrated into the work life yielded interesting results. The participants had a perfect split in their opinions. Exactly as many persons as were agreeing to the statement had the opposite opinion. Thus the mean was 2 ($\sigma = 1.41$) and the distribution was consequently very large (see figure 6.5).

There were two more questions, which are not illustrated with a chart. They participant should evaluate, if the process could improve the onboarding process. This

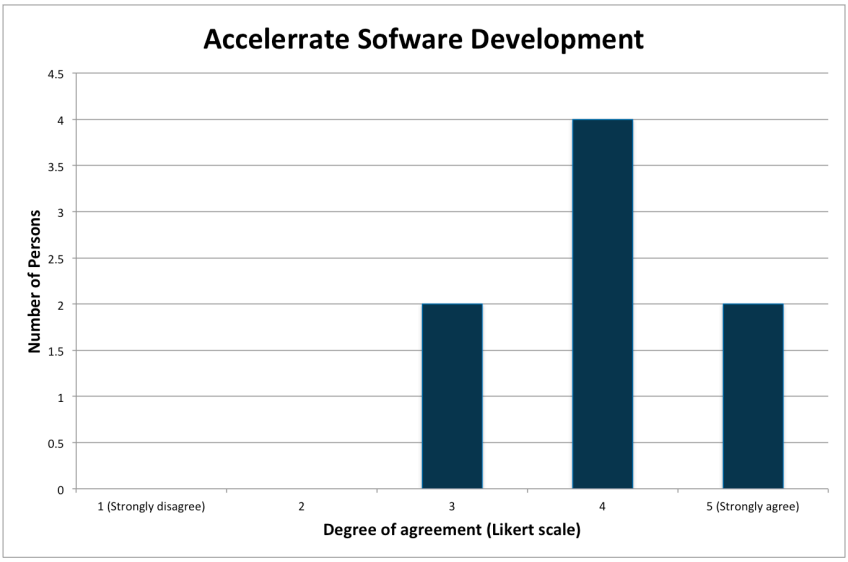


Figure 6.3.: Evaluation of the software development acceleration.

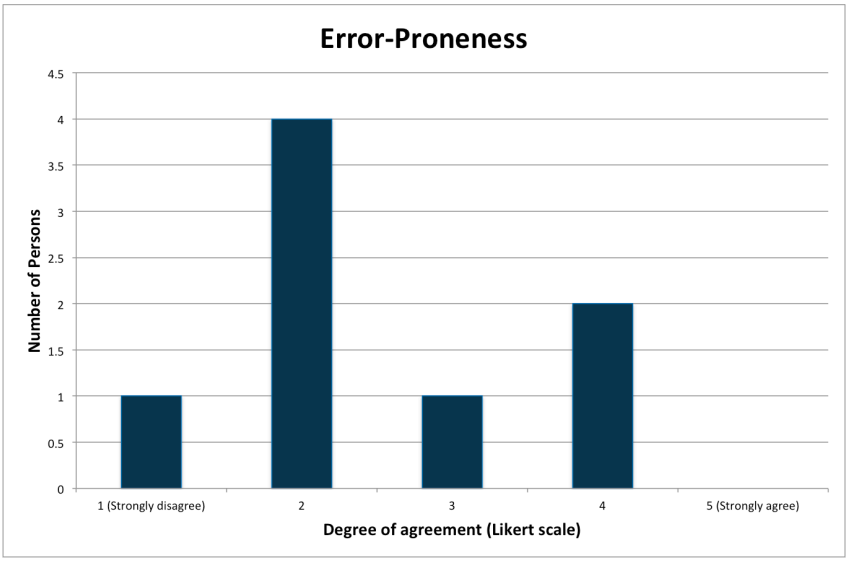


Figure 6.4.: Evaluation of the error-proneness.

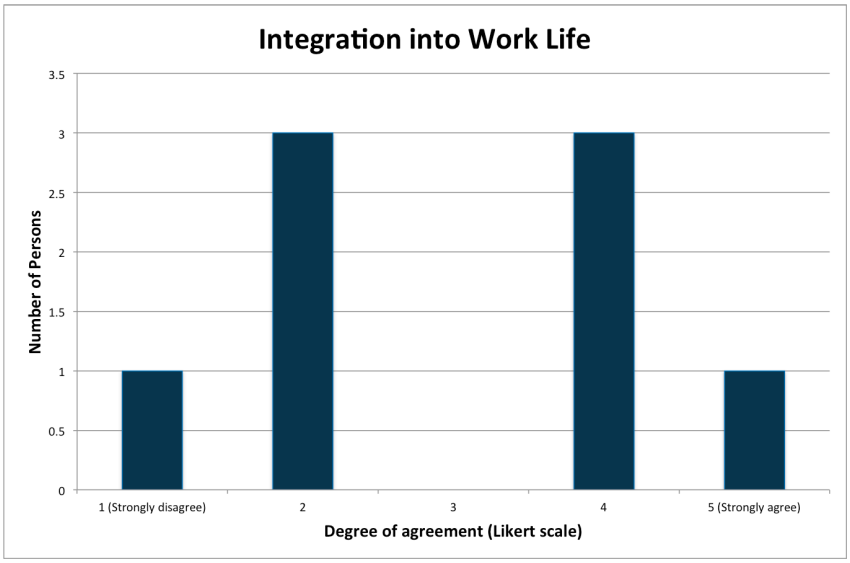


Figure 6.5.: Evaluation of the process' ability to be integrated into the work life.

targeted the verification of the need for a shared component catalog. 37.5% of the participants (3 persons) agreed with the statement. The mean was 2.6 ($\sigma = 1.30$), which indicates a slight overall agreement. The last question about the production readiness of the proposed solution caused comprehension questions by the participants in three cases. Since the statement was obviously formulated ambiguously it is not considered in the course of this evaluation. However, the answer are nevertheless attached in the appendix.

In summary this part of the questionnaire revealed that the key requirements are fulfilled by the designed prototyping process. The majority of participants did not perceive the process and tool as too complex or error-prone. However, the opinions about the integration into ones daily work was split. A possible reason for this result might be the composition of the participant group. Approximately 50% of the participants – namely the team lead(s) and operation manager(s) – might not immediately use the process and thus see no reason for an integration into their work life.

6.2. Usability

The usability of the proposed solution was measured with the System Usability Scale (SUS) and the detailed calculation of the SUS score could be found on the enclosed CD. The average SUS score was 67.19, thus it almost exactly corresponds to the overall average SUS score of 68, which was determined by research (`https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`). Considering the throughout positive impression of the first part it is surprising that the usability score is slightly

below the average. The feedback, which was additionally provided at the end of the questionnaire and collected during the walkthrough could help to better understand this result.

6.3. Feedback

At end of the questionnaire two participants provided additional written feedback. One participant emphasised that the process "would enhance [the] collaboration and speed up [the] software development". The second participant highlighted the benefits of the component library, which was described as "a great tool [improve] transparency and [enable a] better reuse of existing code over team boundaries". The participant also criticised the code generation as potentially being "very complicated an error prone [...]". The original feedback of the questionnaire in its unedited form could be found in section E.3 of the appendix.

The opinion that the developed component modeller could significantly improve the collaboration between departments was verbally expressed by two more participants. The aspect of multi-fidelity was not mentioned by a single person during the walkthrough or feedback of the questionnaire. Even upon request the participants reacted with moderately with the confirmation that multi-fidelity is an interesting feature.

7. Summary

In chapter 3 a broad literature research, covering academic as well as professional resources, was utilised to define the terms fidelity and mockup-driven development. Two major fidelity-levels were identified and common usability artefacts assigned to them. This thesis established the understanding that a prototype could have a low- or high-fidelity level. Thus mockups are an example for a high-fidelity prototype. Furthermore the multi-fidelity prototyping was defined as the systematic process of increasing the fidelity of a view or component by the addition of new properties. The definition of mockup-driven development found in literature did not follow this idea and only covers the transition from a high-fidelity mockup to the final product or was in the case of the *Cascading Tree Sheets* more or less unrelated to the presented concept.

This insight and the core requirements of a collaboration improvement and the systematic reuse of existing components, which were identified together with the industry partner in chapter 2, lead to suggestion of a custom prototyping process. Under the consideration of the established term definitions the process is called *prototype-driven development*. The inclusion of the application's or component's user takes an essential role in the defined process. Furthermore should the systematic specification and design of components improve the communication between departments and speed up the development through a better alignment with the requirement engineers and the facilitation of a repurposing of prototypes for the UI code generation.

During the implementation a software prototype for a multi-fidelity component modeller and catalog was developed. The evaluation confirmed that the focus on this aspect of the whole mockup-driven development process was set correctly and that the developed solution helps to improve the collaboration and reuse of components. However, the last and prominent question, which formed the title of this thesis, is not answered yet. Do multi-fidelity levels improve mockup-driven development? Following Rivero's definition of mockup-driven development in [46] this approach takes care of the transition from a high-fidelity mockup to the final user interface. Reducing the fidelity of the mockup will also reduce the one of the generated product. This approach is difficult to reconcile with the presented multi-fidelity process and the reuse of components. Thus the mockup-driven development process as defined does not profit from a multi-fidelity approach as it requires an annotated high-fidelity artefact as input.

If one views the mockup-driven-development process embedded in the prototype-driven development process as the last step, one has to consider the results of the

previously presented evaluation. No participant identified any benefit in the integration of multiple fidelity-levels. A potential explanation could be that firstly the aspect of multi-fidelity was not motivated by concrete requirements of the industry partner, but identified as a research gap from the academic perspective. Secondly one has to consider the currently established UCD activities, which were described in the problem identification and the two case studies. The definition and introduction of a collaborative UCD process is more urgent then the seamless support of fidelity transitions. Furthermore the existence of such a process could be seen as a requirement for the installation of a multi-fidelity process. This aspect was also considered during the design of the prototyping process and influenced that the process could be applied independently of the fidelity-level.

Finally the results should be reviewed from a critical view point. The results of the evaluation might be criticised, because they did not evaluate the real prototyping process with all its steps, but only present a walkthrough and one aspect – the component modeller – as a concrete solution. However, it was impossible to design and install such a complex process at a large company within the 6 month duration of a master's thesis. Wherever necessary and possible the requirements and steps of the process were prioritised according to the needs of the industry partner and technological requirements. Finally the feedback impressively confirmed that the prioritisation was made correctly. The envisioned process provides room for a variety of future research. The following section is going to present several options as part of the outlook.

8. Outlook and Future Research

Andreas Tielitz presented in his master's thesis an approach for the automatic extraction of view-models from components [54]. This technology could be utilised to analyse a existing component base and to automatically create representations in the component modeller. Following the idea of continuous integration one could even go one step further and enhance the prototyping process by the automatic maintenance of the UI component catalog on the basis of the source code. This would ensure that no more manual efforts are necessary to always keep the catalog up to date.

The defined components have to be made available for the prototyping tool used by the requirements engineer. In the case of Siemens, this would require the development of a custom plugin for the prototyping software *Justinmind*. At this point it might also make sense to connect the tool to the ALM solution, which would enable a direct access to the requirements and allow the requirements engineer to complete his part of the prototyping process within one application.

After the handover of the prototype to the software engineer a code generator needs to parse and analyse the prototype's data structure to identify the utilised components. Then a representation of the prototype in the form of the specified view model has to be made, which subsequently allows the generation of the UI code. The developed component modeller has only very basic support for this approach. Furthermore the "prototype parser" has to be developed for each prototyping tool as they use proprietary data formats (this aspect was also analysed during the thesis and several tools used JSON-based data formats with custom structures).

Finally the idea of multi-fidelity could be reconsidered. A paper-based sketch is also a prototype, which basically uses the tools of pen and paper. It is imaginable to use image recognition to match a sketched component to its low-fidelity counterpart in the component catalog. This would finally enable the envisioned multi-fidelity process. The connection would to the digital component would allow the transition to any other fidelity-level or even to the implementation of the component. Imagine the following scenario: After sketching a user interface one takes a picture of the low-fidelity prototype with a smartphone and uploads it to a cloud service. Image recognition algorithms match the drawn UI components to a existing catalog of components. The system automatically transforms the matched digital representation of the sketch to a realistic software prototype, which could be used as the basis for the development.

List of Figures

List of Tables

Glossary

Application Lifecycle Management (ALM) "[...] The product lifecycle management (governance, development, and maintenance) of computer programs. It encompasses requirements management, software architecture, computer programming, software testing, software maintenance, change management, continuous integration, project management, and release management" [65]. 10

Business Process Management Notation (BPMN) " [...] A graphical notation that depicts the steps in a business process." [41]. 33

Business-to-Consumer (B2C) "The business-to-consumer market is the common form of the market, where enterprises supply the demand of consumers" [50]. 3

Commercial Off-the-Shelf (COTS) "Commercially available specialized software designed for specific applications (such as legal or medical billing, chemical analysis, statistical analysis) that can be used with little or no modification" [8]. 4

Consumerization "The invasion of innovation in the enterprise context, which actually originates from the consumer sector" [64]. 3

Continuous Integration (CI) "[...] A development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early" [53]. 5

Mobile Device Management (MDM) "[...] includes software that provides the following functions: software distribution, policy management, inventory management, security management and service management for smartphones and media tablets. MDM functionality is similar to that of PC configuration life cycle management (PCCLM) tools; however, mobile-platform-specific requirements are often part of MDM suites" [15]. 6

Return on Investment (ROI) "A performance measure used to evaluate the efficiency of an investment or to compare the efficiency of a number of different investments. ROI measures the amount of return on an investment relative to the investments cost" [22]. 4

System Usability Scale (SUS) "[...] A simple, ten-item scale giving a global view of subjective assessments of usability" [6]. 8, 48, 53, 57

User-Centered Design (UCD) . 4, *see* Human-Centered Design (HCD)

Bibliography

[1] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster. "UIML: An appliance-independent XML user interface language." In: *Computer Networks* 31.11 (1999), pp. 1695–1708. ISSN: 13891286. DOI: 10.1016/S1389-1286(99)00044-4.

[2] J. Allen. *So What Exactly is a View-Model?* 2012. URL: https://www.infoq.com/articles/View-Model-Definition.

[3] M. K. Azham Hussain. "Apps vs Devices: Can the Usability of Mobile Apps be Decoupled from the Device?" In: *International Journal of Computer Science Issues* 9.3 (2012), pp. 11–16.

[4] E. Benson. "Mockup Driven Web Development." In: *Proceedings of the 22nd international conference on World Wide Web companion* (2013), pp. 337–341.

[5] R. G. Bias and D. J. Mayhew. *Cost-justifying usability: an update for an Internet age*. Ed. by R. G. Bias and D. J. Mayhew. Vol. Second. 2006-7. Morgan Kaufmann, 2005, p. 687. ISBN: 0120958112. DOI: ISBN-10:0120958112.

[6] J. Brooke. "SUS - A quick and dirty usability scale." Early, 1986.

[7] B. Bruegge and A. H. Dutoit. *Object-oriented software engineering : using UML, patterns and Java*. 2003, p. 762. ISBN: 0130471100.

[8] Business Dictionary. *Commercial Off The Shelf Software Definition*. 2016. URL: http://prod.sandia.gov/techlib/access-control.cgi/2006/060478.pdf (visited on 11/30/2016).

[9] Business Wire. *iRise Joins Microsoft Visual Studio Industry Partner Program*. 2009. URL: http://www.businesswire.com/news/home/20090309005394/en/iRise-Joins-Microsoft-Visual-Studio-Industry-Partner (visited on 12/07/2016).

[10] A. Chen. *Why low-fidelity prototyping kicks butt for customer-driven design*. URL: http://andrewchen.co/why-every-consumer-internet-startup-should-do-more-low-fidelity-prototyping/ (visited on 07/10/2016).

[11] A. Coyette, S. Kieffer, and J. Vanderdonckt. "Multi-fidelity prototyping of user interfaces." In: *Human-Computer Interaction INTERACT* 4662 (2007), pp. 150–164. ISSN: 0302-9743. DOI: 10.1007/978-3-540-74796-3_16.

[12] Design Modo. *The What, Why and How of Mockups*. 2015. URL: http://designmodo.com/mockups/ (visited on 07/10/2016).

[13] J. S. Dumas and J. C. Redish. *A practical guide to usability testing*. Vol. Rev. ed. Intellect, 1999, p. 404. ISBN: 1841500208.

[14] X. Ferré, N. Juristo, H. Windl, and L. Constantine. "Usability basics for software developers." In: *IEEE Software* 18.1 (2001), pp. 22–29.

[15] Gartner. *IT Glossary - Mobile Device Management*. URL: http://www.gartner.com/it-glossary/mobile-device-management-mdm/ (visited on 12/06/2016).

[16] J. Herman. "A process for creating the business case for user experience projects." In: *CHI'04 extended abstracts on Human factors in ...* (2004), pp. 1413–1416.

[17] A. R. Hevner, S. T. March, J. Park, and S. Ram. "Design Science in Information Systems Research." In: *MIS Quarterly* 28.1 (2004), pp. 75–105. ISSN: 02767783. DOI: 10.2307/25148625.

[18] K. Holtzblatt, J. B. Wendell, and S. Wood. *Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design*. Vol. 2005. 2005, p. 320. ISBN: 0123540518. DOI: 10.1145/1066322.1066325. URL: http://books.google.com/books?hl=en{\&}lr={\&}id=VjO6n9stHzUC{\&}pgis=1.

[19] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. "Empirical assessment of MDE in industry." In: *2011 33rd International Conference on Software Engineering (ICSE)* (2011), pp. 471–480. ISSN: 0270-5257. DOI: 10.1145/1985793.1985858.

[20] comScore Inc. *The U.S. Mobile App Report*. Tech. rep. 2014.

[21] Interaction Design. *Interaction Design Foundation: Mockup Definition*. URL: https://www.interaction-dedsign.org/literature/book/the-glossary-of-human-computer-interaction/mock-ups (visited on 07/11/2016).

[22] Investopedia. *Return on Investment - ROI*. (Visited on 12/06/2016).

[23] Justinmind. *Integration with Atlassian JIRA: importing and exporting JIRA issues in Justinmind Enterprise*. 2016. URL: https://www.justinmind.com/support/integration-with-atlassian-jira-importing-and-exporting-jira-issues-in-justinmind-enterprise/ (visited on 12/07/2016).

[24] K. Kidd. *iRise provides a way for users to test-drive Web-based software*. 2004. URL: http://www.techrepublic.com/article/irise-provides-a-way-for-users-to-test-drive-web-based-software/ (visited on 12/07/2016).

[25] K. Kohler and T. Hochreuter. "Let's compare prototypes for tangible systems." In: *Proceedings of the 8th Nordic Conference on Human-Computer Interaction Fun, Fast, Foundational - NordiCHI '14*. New York, New York, USA: ACM Press, 2014, pp. 323–332. ISBN: 9781450325424. DOI: 10.1145/2639189.2639229. URL: http://dl.acm.org/citation.cfm?id=2639189.2639229.

[26] A Lancaster. "Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces." In: *Ieee Transactions On Professional Communication* 47.4 (2003), pp. 335–336.

[27] Y.-K. Lim, A. Pangam, S. Periyasami, and S. Aneja. "Comparative analysis of high- and low-fidelity prototypes for more valid usability evaluations of mobile devices." In: *Proceedings of the 4th Nordic conference on Human-computer interaction changing roles - NordiCHI '06* October (2006), pp. 291–300. ISSN: 1595933255. DOI: 10.1145/1182475.1182506.

[28] Y.-K. Lim, E. Stolterman, and J. Tenenberg. "The anatomy of prototypes." In: *ACM Transactions on Computer-Human Interaction* 15.2 (2008), pp. 1–27. ISSN: 10730516. DOI: 10.1145/1375761.1375762.

[29] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero. "USIXML: A Language Supporting Multi-path Development of User Interfaces." In: *Ehci/Ds-Vis* (2005), pp. 200–220. ISSN: 03029743. DOI: 10.1007/11431879_12.

[30] A. Marcus. "User-centered design in the enterprise." In: *interactions* 12.1 (2005), p. 18. ISSN: 10725520. DOI: 10.1145/1041280.1041293.

[31] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, and A. Vera. "Breaking the Fidelity Barrier - An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success." In: *Proceedings of the International Conference on Human Factors in Computing Systems (CHI'06)* (2006), pp. 1233–1242. DOI: 10.1145/1124772.1124959.

[32] T. Memmel, F. Gundelsweiler, and H. Reiterer. "Agile Human-Centered Software Engineering." In: *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it - Volume 1* (2007), pp. 167–175. ISSN: 0040-5736. DOI: 10.1177/004057368303900411.

[33] T. Memmel, F. Gundelsweiler, and H. Reiterer. "Prototyping Corporate User InterfacesTowards A Visual Specification Of Interactive Systems." In: *Proceedings of the Second IASTED International Conference on Human-Computer Interaction*. 2007, pp. 177–182. ISBN: 9780889866546 (ISBN). URL: http://www.actapress.com/PDFViewer.aspx?paperId=30110.

[34] M. Meuser and U. Nagel. "The expert interview and changes in knowledge production." In: *Interviewing Experts*. 2009, pp. 17–42. ISBN: 9780230220195. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[35] J. Meyer. *Creating Informal Looking Interfaces*. 2005. URL: http://www.cybergrain.com/tech/pubs/lines{_}technote.html (visited on 08/05/2016).

[36] Microsoft - User eXperience. *Lo-Fi or Hi-Fi mockups? Blend Ressource Dictionaries mean not having to chose*. 2008. URL: https://blogs.msdn.microsoft.com/shanemo/2008/03/15/lo-fi-or-hi-fi-mockups-blend-resource-dictionaries-mean-not-having-to-choose/ (visited on 07/10/2016).

[37] R. Milzarek. "Analysis of the added value of innovative usability and dynamic visualization concepts for the graphical representation of Siemens AGs KPIs and prototypical implementation of a native iOS-application." PhD thesis. Technical University Munich, 2013.

[38] P. Mohagheghi, M. A. Fernandez, J. A. Martell, M. Fritzsche, and W. Gilani. "MDE adoption in industry: Challenges and success criteria." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5421. 2009, pp. 54–59. ISBN: 9783642016479. DOI: 10.1007/978-3-642-01648-6_6.

[39] F. Morais and A. R. da Silva. "Assessing the Quality of User-Interface Modeling Languages." Lisbon, 2015.

[40] U. movement. *4 Things No One Told Me About High-Fidelity Wireframes*. URL: http://uxmovement.com/wireframes/4-things-no-one-told-me-about-high-fidelity-wireframes/ (visited on 07/10/2016).

[41] Object Management Group Inc. *Business Process Modeling Notation - Specification*. URL: http://www.bpmn.org.

[42] Online Oxford English Dictionary. *Definition Fidelity*. URL: https://en.oxforddictionaries.com/definition/fidelity (visited on 12/07/2016).

[43] F. Paternò, C. Santoro, and L. D. Spano. "MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments." In: *ACM Transactions on Computer-Human Interaction* 16.4 (2009), pp. 1–30. ISSN: 10730516. DOI: 10.1145/1614390.1614394.

[44] C. Phanouriou. "UIML: A Device-Independent User Interface Markup Language." Dissertation. Virginia Polytechnic Institute and State University, 2000.

[45] P. Pinheiro, D. Silva, and N. W. Paton. "UMLi: The Unified Modeling Language for Interactive Applications." In: *International Conference on the Unified Modeling Language*. 2000, pp. 117–132.

[46] J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, F. Montero, and M. Gaedke. "Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering." In: *Information and Software Technology* 56.6 (2014), pp. 670–687. ISSN: 09505849. DOI: 10.1016/j.infsof.2014.01.011.

[47] J. M. Rivero, G. Rossi, J. Grigera, J. Burella, E. R. Luna, and S. Gordillo. "From mockups to user interface models: An extensible model driven approach." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6385 LNCS. 2010, pp. 13–24. ISBN: 3642169848. DOI: 10.1007/978-3-642-16985-4_2.

[48] R. Sefelin, M. Tscheligi, and V. Giller. "Paper prototyping - what is it good for?: a comparison of paper- and computer-based low-fidelity prototyping." In: *CHI'03 extended abstracts on Human factors in computing systems. ACM* (2003), pp. 778–779. ISSN: 1581136374. DOI: 10.1145/765891.765986.

[49] Siemens AG. *Earnings Release Q4 FY 2016*. Tech. rep. Munich, 2016.

[50] Springer Gabler Verlag. *Business-to-Consumer-Markt*.

[51] G. Susan Moore. *Closing the User Experience Gap*. 2015. URL: http://www.gartner.com/smarterwithgartner/closing-the-user-experience-gap/ (visited on 11/29/2016).

[52] G. Susan Moore. *Gartner Says Demand for Enterprise Mobile Apps Will Outstrip Available Development Capacity Five to One*. 2015. URL: http://www.gartner.com/newsroom/id/3076817 (visited on 11/29/2016).

[53] ThoughtWorks. *Continuous Integration Definition*. URL: https://www.thoughtworks.com/continuous-integration (visited on 12/05/2015).

[54] A. Tielitz. "Automatically extracting view models from component-based web applications." Master's Thesis. Technical University of Munich, 2016.

[55] Usability Geek. *Smart UX: High-Fidelity Wireframes*. 2016. URL: http://usabilitygeek.com/smart-ux-highŋ-fidelity-wireframes/ (visited on 07/10/2016).

[56] Usability Geek. *When To Prototype, When To Wireframe - How Much Fidelity Can You Afford?* 2014. URL: http://usabilitygeek.com/when-to-prototype-when-to-wireframe-fidelity/ (visited on 07/10/2016).

[57] Use Tree. *Mockup, Wireframe, Prototyp - was, wann und wie?* 2014. (Visited on 07/10/2016).

[58] UXPin. *Why Designers Shouldn't Neglect Mockups*. URL: https://studio.uxpin.com/blog/designers-shouldnt-neglect-mockups/ (visited on 07/10/2016).

[59] J. Vanderdonckt. "Model-Driven Engineering of User Interfaces : Promises , Successes , Failures , and Challenges." In: *Proceedings of the National Conference on Human-Computer Interaction*. 2008, pp. 1–10. URL: http://rochi.utcluj.ro/rrioc/en/rochi2008.html{\#}Model{_}Driven{_}Engineering.

[60] L. Vernazza. "Himalia: Model-driven user interfaces using hypermedia, controls and patterns." In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Vol. 10. PART 1. 2007, pp. 477–482. ISBN: 9783902661371. DOI: http://dx.doi.org/10.3182/20070904-3-KR-2922.00085.

[61] R Virzi, J. L. Sokolov, and D Karis. "Usability problem identification using both low- and high-fidelity prototypes." In: *Proceedings of the SIGCHI conference on Human factors in computing systems common ground* (1996), pp. 236–243. DOI: 10.1145/238386.238516.

[62] R. a. Virzi, J. L. Sokolov, and D. Karis. "Usability problem identification using both low- and high-fidelity prototypes." In: *Proceedings of the SIGCHI conference on Human factors in computing systems common ground*. 1996, pp. 236–243. ISBN: 0897917774. DOI: 10.1145/238386.238516.

[63] M. Walker, L. Takayama, and J. a. Landay. "High-Fidelity or Low-Fidelity, Paper or Computer? Choosing Attributes when Testing Web Prototypes." In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 46.5 (2002), pp. 661–665. ISSN: 1071-1813. DOI: 10.1177/154193120204600513.

[64] F. Weiß and J. M. Leimeister. *Consumerization: Herausforderungen für das betriebliche Informationsmanagement durch iPhone und Co*. de. 2013. URL: https://www.alexandria.unisg.ch/224109/1/JML{_}434.pdf.

[65] Wikipedia. *Application lifecycle management*. 2016. URL: https://en.wikipedia.org/wiki/Application{_}lifecycle{_}management (visited on 12/06/2016).

[66] C. E. Wilson and S. Rosenbaum. *Cost-Justifying Usability*. Elsevier, 2005, pp. 215–263. ISBN: 9780120958115. DOI: 10.1016/B978-012095811-5/50008-0.

[67] A.-U.-H. Yasar. "Enhancing experience prototyping by the help of mixed-fidelity prototypes." In: *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology - Mobility '07* (2007), p. 468. DOI: 10.1145/1378063.1378137.

Appendix

A. Evaluation of the Semi Structured Interview

A guideline for a semi structured interview was developed and tested with one subject. The test run did not indicate any problems of understanding or with the planned time frame of 30 minutes. The semi structured interview was conducted with $N = 11$ persons of different professional and organisational backgrounds. Due to the setting of the thesis the majority (72.7%) of the interviewees were employees of the industry partner. 6 different roles were covered by the interview and ensure the representativity of the results. It should be noted that the interviewed requirements engineers also have the role of software testers, but spent the majority of their working time in the role of an requirements engineer.

A.1. General Information

a) What is the size of your company?

Company size	Number of persons
Micro (1 - 9 employees)	1
Small (10 - 49 employees)	1
Medium (50 - 249 employees)	1
Large ($\geq$ 250 employees)	8

Table A.1.: Evaluation of the company size.

b) What role do you have within the company?

Role	Number of persons
Software Developer	3
Team Lead	3
Requirements Engineer	2
Demand Manager	1
Management	1
Operations Manager	1

Table A.2.: Evaluation of role within the company.

c) How many years of professional experience do you have?

Years of professional experience	Number of persons
0	1
1	1
8	1
10	2
15	1
20	3
23	1

Table A.3.: Evaluation of the years of professional experience.

A.2. User Centered Design

a) Did you apply user-centered design methods in your professional life?

Applied UCD methods	Number of persons
Yes	9
No	2

Table A.4.: Evaluation of application of user-centered design approaches.

b) Which user-centered design methods did you apply?

- Low-Fidelity sketches (1)
- Low-Fidelity wireframe (1)
- High-fidelity mockup (9)
- High-fidelity software prototypes (3)
- Usability walkthrough (1)
- User interviews (3)
- Persona (2)
- Controlled study in usability lab (1)

Comments:

- Prefer interviews if the user/customer's occupation is not IT-related

c) At what point in time of the project were the user-centered design methods applied?

- Requirement Elicitation (10)
- During the development (5)
- Evaluation of the developed solution (2)

Comments:

- The later the less usage of UCD method
- In agile projects during the whole development
- For internal development projects mostly at the beginning of the project
- Not yet used for comparing external software vendors – respectivley their software solutions

- Provided by the designers as a specification document

d) What was the purpose / rationale behind applying a specific user-centered design method?

- Introduce the Corporate Design (1)
- Collect early feedback about the user interface (1)
- Avoid changes of the requirements late in project (2)
- Speak the same language through discussion and visualisations (4)
- Better understand the problem and process (4)
- Evaluate if the software solves the problem (2)
- Measure the usability (1)
- Inspiration (1)
- Create trust (1)
- Support the project planning (1)
- Support the internal team communication (2)
- Expectation management (2)
- Comprehensibility for decision makers (2)

Comments:

- Identify features which are more stable, group and prioritise them accordingly
- Find a focus for someones own development project. This comment was not considered as it was very specific to the role of a freelance developer working on custom projects

e) Is the application of user-centered design methods defined or regulated by a project execution / software development guideline? Are they integrated into the process?

Established UCD process	Number of persons
Yes	1
Partially	1
No	9

Table A.5.: Evaluation of establishment of a UCD process.

f) How was your personal experience during the application of the user-centered design method?

- Predominantly positive experiences (7)
- Inspiring and creative (2)
- Better understanding of the user's needs (3)
- Easier to comprehend for the customer compared to classical requirement documents (1)
- Additional time effort justified by its benefits (3)
- Better planning (1)
- Very simple methodology (1)
- Better foundation for the communication (1)
- Expectations clarified (1)
- Fast feedback (1)
- Might cause too high expectations (reality vs. mockup) (2)
- Not considering boundary conditions (1)
- No UI component templates available (1)
- Content of the prototypes must be tailored to the target group (experts expect realistic data) (1)
- Difference between customer and user often neglected (1)
- Management might be hard to convince of the benefits, because the primarily see additional costs (1)

Comments:

- Advantages could not be evaluated properly, as the interviewee was not involved in the later stages of the project
- Usability expert separate from the requirements engineer gave a new perspective and valuable feedback

- Use the falsification principle, which requires you to know what you want to measure or analyse
- Mockups are usually not a green field approach and have to consider which UI elements could be reused and what is technically feasible
- Disposable mockups should be avoided!
- Possibility to use mockups to create video guides for the manual tests
- One additional method of requirement elicitation (use cases, mockupts, etc.)
- A cohesive style guide instead of multiple mockups/design could increase the development speed and assure a consistent look and feel

B. Degree of Fidelity Analysis

For the analysis of the different degrees of fidelity a comparison between a plain button element was made. Therefore a sketched button (figure B.1), *Justinmind's* representation of a button (figure B.2) and the button element of the *Twitter Bootstrap* (figure B.3) and *Material Design* (figure B.4) framework were compared.

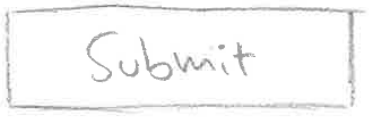


Figure B.1.: Sketched button

Figure B.2.: *Justinmind* button

Figure B.3.: Button of *Twitter Bootstrap*

Figure B.4.: Button of *Material Design*

In table B.1 the number of stylesheet attributes, which were necessary to create the previous visualisations are listed. For the sketch the attributes were defined by the author and thus might be subjective. However, even if the actually necessary number of attributes is twice as large, this does not alter the trend. The attributes for the prototyping tool *Justinmind* were derived from the generated HTML code of the mockup.

	Sketch	Justinmind	Twitter Bootstrap	Angular Material
	position	position	color	color
	height	top	background-color	background-color
	width	left	border-color	background-image
	bottom	width	display	background-position-x
	right	height	padding-top	background-position-y
	left	background-color	padding-right	background-size
	top	background-image	padding-bottom	background-repeat-x
		border-top-color	padding-left	background-repeat-y
		border-top-style	margin-bottom	background-attachment
		border-top-width	font-size	background-origin
		border-left-color	font-weight	background-clip
		border-left-style	line-height	box-shadow
		border-left-width	text-align	border-top-color
		border-bottom-color	white-space	border-top-style
		border-bottom-image	vertical-align	border-top-width
		border-bottom-style	touch-action	border-left-color
		border-right-color	cursor	border-left-style
		border-right-style	user-select	border-left-width
		border-right-width	background-image	border-bottom-color
		border-top-left-radius	border-top-style	border-bottom-image
		border-top-right-radius	border-top-width	border-bottom-style
		border-bottom-left-radius	border-left-style	border-right-color
		border-bottom-right-radius	border-left-width	border-right-style
		padding-top	border-bottom-image	border-right-width
		padding-right	border-bottom-style	border-top-left-radius
		padding-bottom	border-right-style	border-top-right-radius
		padding-left	border-right-width	border-bottom-left-radius
		transform	border-top-left-radius	border-bottom-right-radius
		box-shadow	border-top-right-radius	position
		...	...	...
∑	**7**	**37**	**42**	**71**

Table B.1.: Comparison of the number of CSS attributes between different button representations

C. Prototyping Process Charts

On the following two pages you find an enlarged version of the prototype-driven development process, which was presented in figure . Furthermore the original and a digital version of the chart could be found on the enclosed CD.

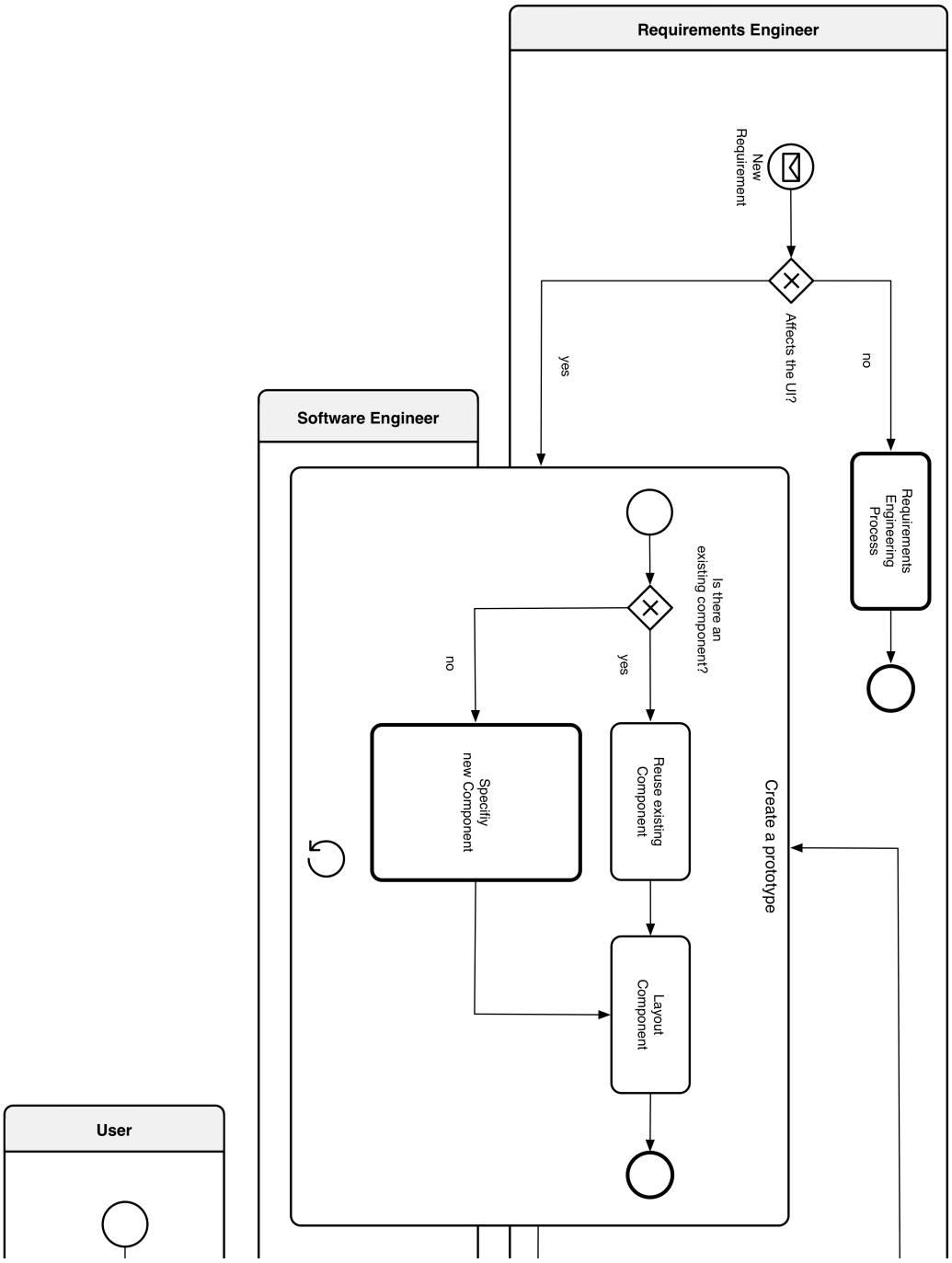


Figure C.1.: The prototype-driven development process (enlarged part 1).

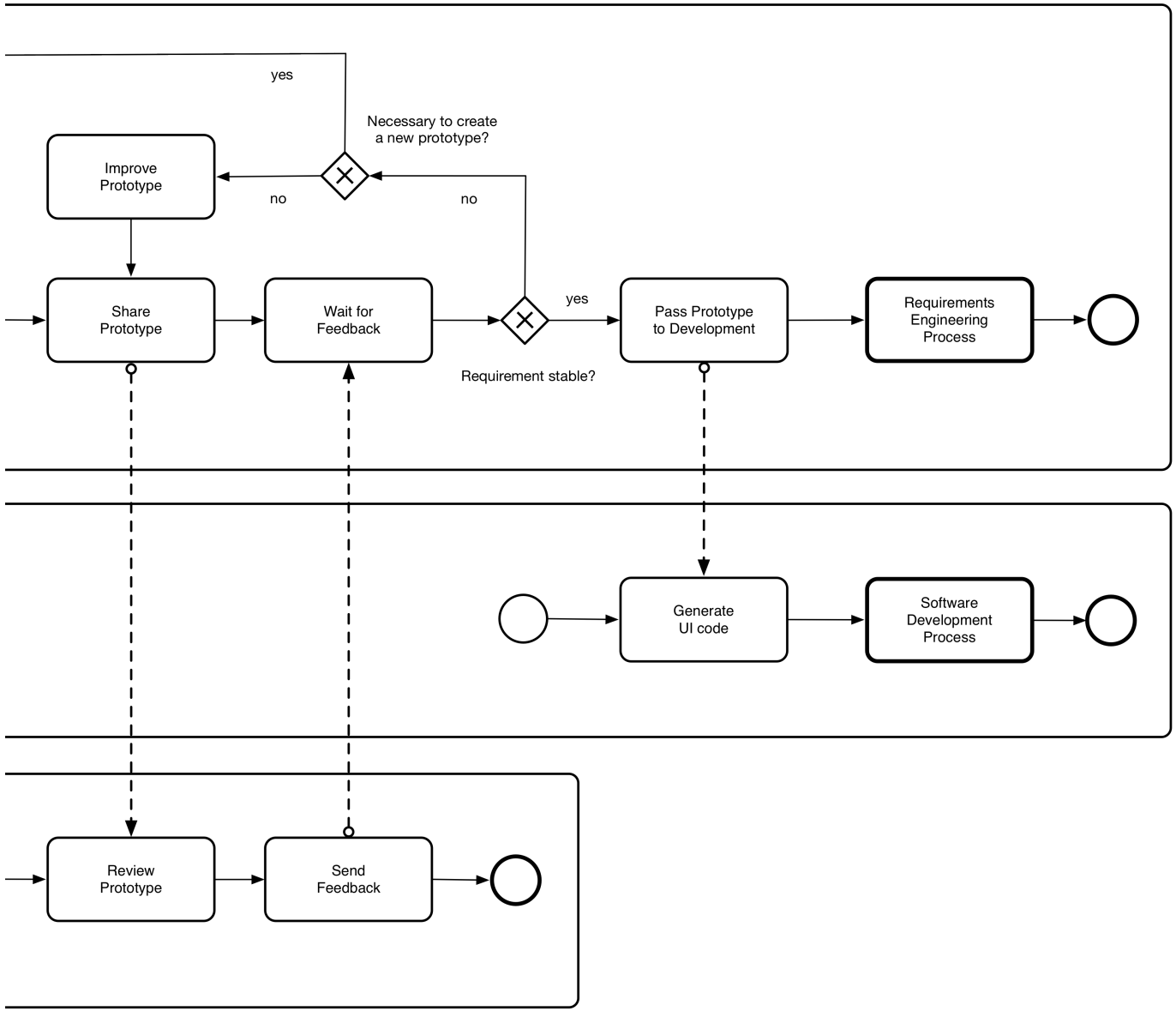


Figure C.2.: The prototype-driven development process (enlarged part 2).

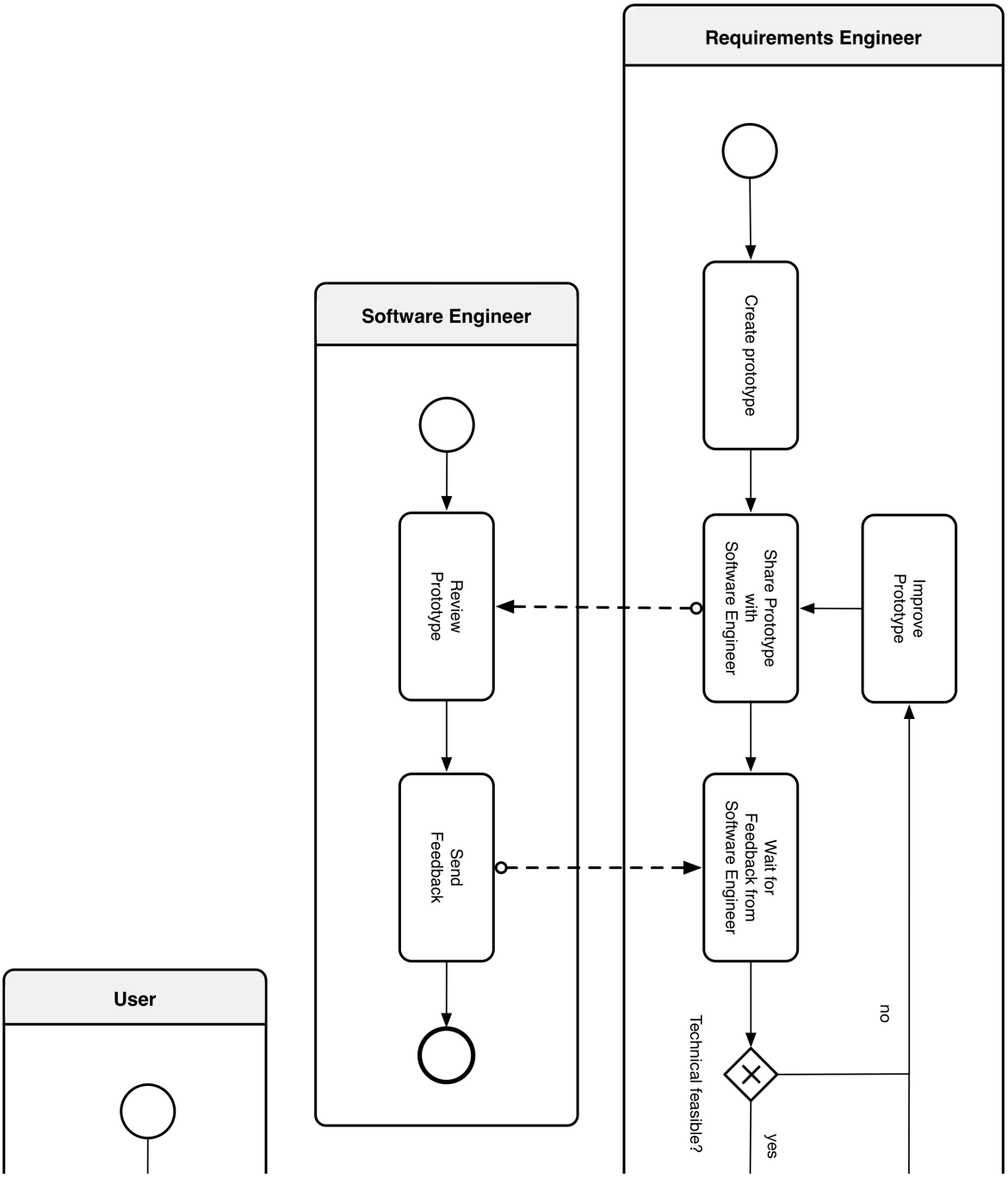


Figure C.3.: The subprocess for the specification of a new component (enlarged part 1).

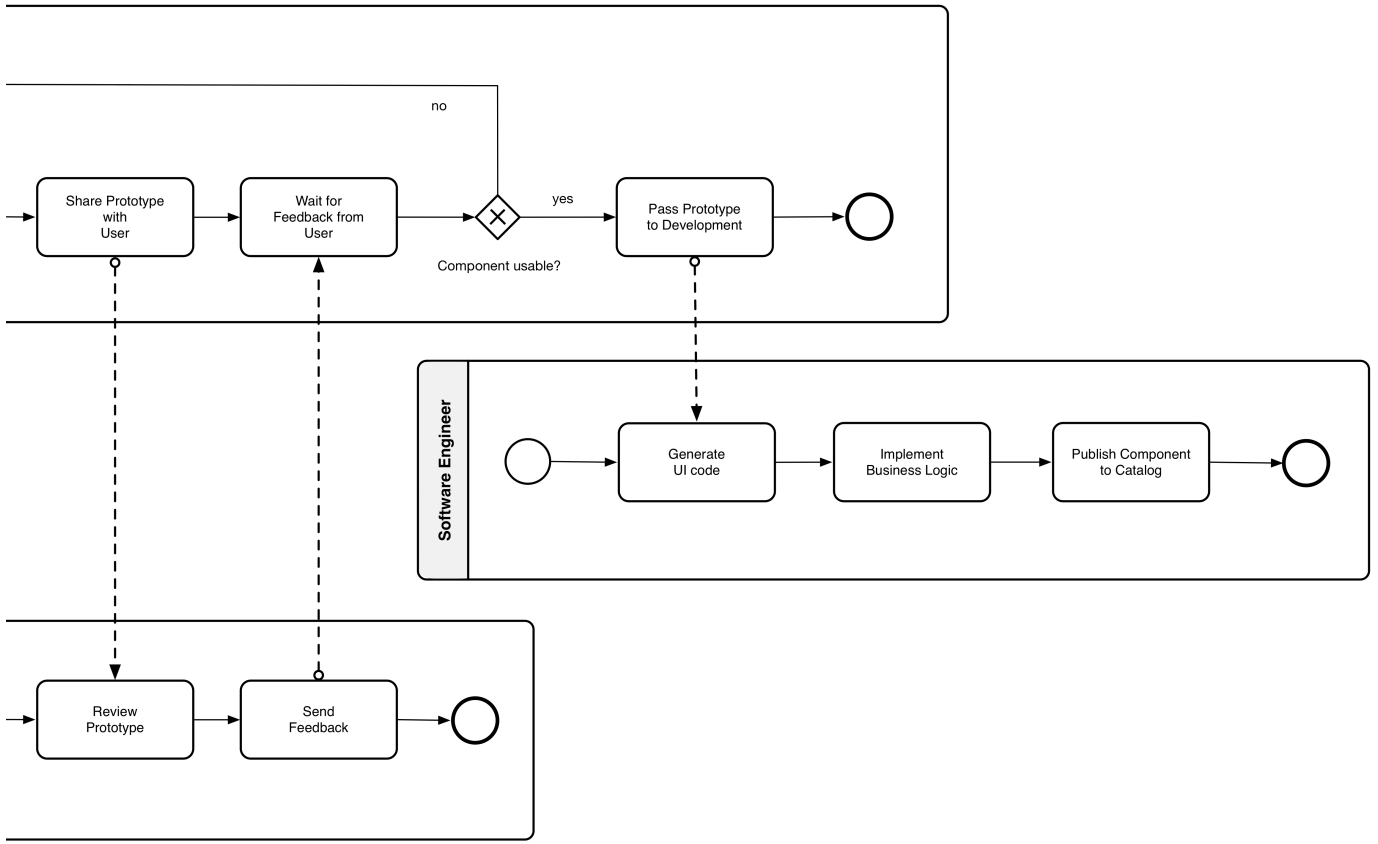


Figure C.4.: The subprocess for the specification of a new component (enlarged part 2).

D. Third Party Libraries

Library	License	Website
AngularJs	MIT license	https://github.com/angular/angular
Angular Material Design	MIT license	https://github.com/angular/material
Angular UI Router	MIT license	https://github.com/angular-ui/ui-router
Angular UI Tree	MIT license	https://github.com/angular-ui-tree/angular-ui-tree
Angular NVD3	MIT license	https://github.com/krispo/angular-nvd3
NVD3	Apache License, Version 2.0	https://github.com/novus/nvd3
Angular UI Codemirror	MIT license	https://github.com/angular-ui/ui-codemirror
Codemirror	MIT license	https://github.com/codemirror/CodeMirror
Fontawesome	MIT license, SIL OFL 1.1 and CC BY 3.0	https://github.com/FortAwesome/Font-Awesome

Table D.1.: Comparison of prototyping tools according to the specified requirements.

E. Evaluation of the Online Questionnaire

The questionnaire was prepared with *Lime Survey* (`https://www.limesurvey.org`) and hosted on the author's webspace. In advance to the questionnaire a usability walk-through, which explained the prototype-driven development process, took place. Overall $N = 8$ employees participated and 100% completed the survey.

E.1. Suitability

a) I think that the system could increase the collaboration between departments.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	1	12.50%
3	0	0.00%
4	1	12.50%
Strongly agree 5	6	75.00%
No answer	0	0.00%

Table E.1.: Evaluation of the collaboration improvement.

b) I think that the system enhances the reuse of existing components.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	1	12.50%
3	0	0.00%
4	1	12.50%
Strongly agree 5	6	75.00%
No answer	0	0.00%

Table E.2.: Evaluation of the reuse enhancement.

c) I think that the system could accelerate the software development.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	0	0.00%
3	2	25.00%
4	4	50.00%
Strongly agree 5	2	25.00%
No answer	0	0.00%

Table E.3.: Evaluation of the software development acceleration.

d) I think that the system is prone to errors (fehleranfällig).

Answer	Count	Percentage
Strongly disagree 1	1	12.50%
2	4	50.00%
3	1	12.50%
4	2	25.00%
Strongly agree 5	0	0.00%
No answer	0	0.00%

Table E.4.: Evaluation of the error-proneness.

e) I think that the system could be easily integrated in my everyday work.

Answer	Count	Percentage
Strongly disagree 1	1	12.50%
2	3	37.50%
3	0	0.00%
4	3	37.50%
Strongly agree 5	1	12.50%
No answer	0	0.00%

Table E.5.: Evaluation of the ability to be integrated into the everyday work.

f) I think that the system improves the onboarding of new employees.

Answer	Count	Percentage
Strongly disagree 1	1	12.50%
2	0	0.00%
3	2	25.00%
4	3	37.50%
Strongly agree 5	2	25.00%
No answer	0	0.00%

Table E.6.: Evaluation of the onboarding improvement.

g) I think that the system is not ready for production use.

Answer	Count	Percentage
Strongly disagree 1	2	25.00%
2	3	37.50%
3	0	0.00%
4	2	25.00%
Strongly agree 5	1	12.50%
No answer	0	0.00%

Table E.7.: Evaluation of the production readiness.

E.2. Usability

a) I think that I would like to use this system frequently.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	1	12.50%
3	2	25.00%
4	4	50.00%
Strongly agree 5	1	12.50%
No answer	0	0.00%

Table E.8.: Evaluation of the frequency of use.

b) I found the system unnecessarily complex.

Answer	Count	Percentage
Strongly disagree 1	1	12.50%
2	4	50.00%
3	0	0.00%
4	2	25.00%
Strongly agree 5	1	12.50%
No answer	0	0.00%

Table E.9.: Evaluation of the system's complexity.

c) I thought the system was easy to use.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	1	12.50%
3	2	25.00%
4	1	12.50%
Strongly agree 5	4	50.00%
No answer	0	0.00%

Table E.10.: Evaluation of the ease of use.

d) I think that I would need the support of a technical person to be able to use this system.

Answer	Count	Percentage
Strongly disagree 1	2	25.00%
2	2	25.00%
3	1	12.50%
4	3	37.50%
Strongly agree 5	0	0.00%
No answer	0	0.00%

Table E.11.: Evaluation of the need for technical support.

e) I found the various functions in this system were well integrated.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	1	12.50%
3	1	12.50%
4	4	50.00%
Strongly agree 5	2	25.00%
No answer	0	0.00%

Table E.12.: Evaluation of the integration of the system's functions.

f) I thought there was too much inconsistency in this system.

Answer	Count	Percentage
Strongly disagree 1	4	50.00%
2	1	12.50%
3	1	12.50%
4	2	25.00%
Strongly agree 5	0	0.00%
No answer	0	0.00%

Table E.13.: Evaluation of the system's inconsistency.

g) I would imagine that most people would learn to use this system very quickly.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	2	25.00%
3	0	0.00%
4	4	50.00%
Strongly agree 5	2	25.00%
No answer	0	0.00%

Table E.14.: Evaluation of the learnability.

h) I found the system very cumbersome to use.

Answer	Count	Percentage
Strongly disagree 1	4	50.00%
2	1	12.50%
3	1	12.50%
4	2	25.00%
Strongly agree 5	0	0.00%
No answer	0	0.00%

Table E.15.: Evaluation of the fussiness.

i) I felt very confident using the system.

Answer	Count	Percentage
Strongly disagree 1	0	0.00%
2	1	12.50%
3	1	12.50%
4	4	50.00%
Strongly agree 5	2	25.00%
No answer	0	0.00%

Table E.16.: Evaluation of the user's confidence.

j) I needed to learn a lot of things before I could get going with this system.

Answer	Count	Percentage
Strongly disagree 1	3	37.50%
2	1	12.50%
3	1	12.50%
4	2	25.00%
Strongly agree 5	1	12.50%
No answer	0	0.00%

Table E.17.: Evaluation of the amount necessary knowledge to collect before using the system.

E.3. Feedback

Furthermore 3 persons (37.50%) provided additional feedback, although one only entered “no”.

- I think the first part of the component library is a great tool this will help for transparency and better reuse of existing code over team boundaries. But the code generation will be a very complicated an error prone part. Data handling and quick changes to the mock up will be harder. Personally i would recommend to start creating the first part to have the technology oriented components in a library and if possible a generator for the components to use in tools like just in mind. The code generation for the developer i din’t see as needed as here the requirements for a clean mock up are to high. Mock up should be quick and easy and not have to much dependencies.
- no
- nice idea and good framework/system, would enhance collaboration and speed up software development!