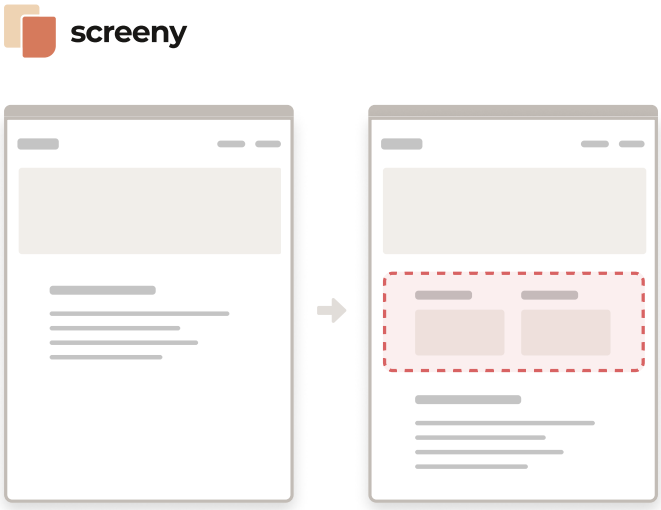
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Facilitation of Regular Communication between UI Designers and Developers through a Continuous Pipeline Tool

A Design Science Study

Master's thesis in Computer science and engineering

JESPER LINDSTRÖM
LUKA MRKONJIC

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Facilitation of Regular Communication between UI Designers and Developers through a Continuous Pipeline Tool

A Design Science Study

JESPER LINDSTRÖM
LUKA MRKONJIC

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Facilitation of Regular Communication between UI Designers and Developers
A Design Science Study
JESPER LINDSTRÖM
LUKA MRKONJIC

Supervisor: Regina Hebig, Computer Science and Engineering
Advisor: Johan Frej, Intunio AB
Examiner: Michel Chaudron, Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of the detection of UI changes, as seen in the artifact.

Typeset in LaTeX
Gothenburg, Sweden 2020

Facilitation of Regular Communication between UI Designers and Developers
A Design Science Study
JESPER LINDSTRÖM
LUKA MRKONJIC
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As a result of ever-increasing user expectations on software, UI designers have a vital role to play in software development projects. Modern software is often built by cross-functional teams consisting of both engineering and design expertise. As a result, the number of UI designers for every developer has increased drastically in the past years, signaling their growing influence. However, the collaboration between the two disciplines comes with its challenges.

This study aims to improve the collaboration between designers and developers during the UI implementation process. An iterative Design Science Research methodology was used to understand the problem, propose a solution, and evaluate its effectiveness. Data was collected in collaboration with our industry partner, Intunio, using a variety of methods, including a focus group, a questionnaire, and several interviews.

We identify a total of six problems that relate to the process of implementing user interfaces. Most notably, we find that UI designers are often not fully aware of the UI implementation progress, which causes design errors to be detected late in the process. Furthermore, the two disciplines were hesitant to initiate communication in order not to disturb the other person. In an attempt to mitigate a selection of the problems, we propose a novel software artifact, Screeny, that facilitates regular communication between designers and developers. The intention is to catch design errors as early as possible. Furthermore, the solution enables the designers to track the implementation progress of the UI.

The evaluation of the artifact was performed using interviews in conjunction with an artificial simulation approach. The data indicated that the solution might successfully mitigate the problems to some degree. In particular, the solution appears to shorten the feedback loop and reduce the barrier of contact between designers and developers.

Keywords: cross-functional teams, designer-developer collaboration, design handoff, design breakdowns, design science research, automated gui testing, visual regression testing, software artifact, screeny.

Acknowledgements

First and foremost, we would like to thank our academic supervisor Regina Hebig and our industry supervisor Johan Frej for all the guidance we have received throughout this period. We would also like to thank all the nice Intunio employees for participating in our study and for making us feel at home in the workplace during the past months. Furthermore, we would like to thank our examiner Michel Chaudron for all the thoughtful feedback and Eric Knauss for publishing helpful guidelines that brought our thesis on the right track. Finally, we would like to extend our gratitude to our family and friends for their moral support.

Jesper Lindström and Luka Mrkonjic, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

As a result of ever-increasing user expectations on software, modern software development is often conducted by cross-functional teams, consisting of both engineering and design expertise. For instance, the number of user interface designers (hereafter denoted "designers") at large companies has increased tremendously in relation to developers. DeAmicis (2019) states that Atlassian went from having one designer for every 25 developers in 2012 to one for every nine in 2017. Most notably, during the same five year period, IBM went from having a ratio of one to 72, to one to eight.

As noted by DeAmicis (ibid.), users have become used to well crafted digital experiences on mobile devices, which increased the expectations for all other kinds of user interfaces (UI) as well. Thus, designers have become an increasingly important part of the product development process. However, the collaboration between developers and designers is often far from perfect.

Even though many modern software development teams follow an iterative agile development process, the design phase often precedes the development phase, as it serves as a blueprint of the user interface to implement. Hence, developers and designers may work in isolation, apart from the design hand-off, in which the developer is provided with the final version of the design work in the form of a static visual document portraying the final look of the user interface. However, Ben Nadel (2020) notes that this is a common mistake and suggests that design hand-offs instead should be treated as an ongoing collaborative process rather than a one-time procedure, though this can be hard to achieve. Due to the lack of continuous collaboration, the implementation may diverge from the design, either by mistake or misinterpretation of a vague design specification. Developers may also be forced to make their own design decisions as a result of changed requirements that are not covered by an updated design. The host company, Intunio, agrees that this is a common problem among software development teams.

In this study, we investigate how designers and developers can be brought closer to each other throughout the entire development process, in order to catch errors earlier and increase the overall quality of the implemented interface design. The work was carried out in collaboration with the previously mentioned company, Intunio, a design and development consultancy firm located in Gothenburg, Sweden.

1.1 Statement of the problem

The current design hand-off procedure has its drawbacks. Avoiding errors in the user interface is not trivial, as the design specification is not a perfect representation of the final software, for instance, due to resource limitations. As noted by Maudet et al. (2017), edge cases can occur as the designer may only design one desktop version and one mobile version, leaving all other screen sizes open for interpretation. Therefore, developers often need to make their own design decisions or request help from the designer, where the design is lacking.

As experienced by our industry contact Intunio, various aspects of the final software may be missing or only vaguely communicated in the design specification, such as the previously mentioned variety of screen sizes and different states of the application. The insufficient specification leaves room for interpretation by the developer, causing decisions to be made during the implementation phase, at which the designer may not be present. Hence, the designer may not be aware of the new design decisions made, which may be problematic, assuming the developer lacks design expertise. Ferreira, Sharp, et al. (2011) reports that developers even were reluctant to make design improvisations on their own as they knew by past experiences that such decisions resulted in having to rework the solutions as soon as the designer noticed, leading to wasted effort. Even though tools exist that do try to "bridge the gap" and help in communicating the specification more clearly, they do not fully eliminate the need for developer design decisions since the design is not a perfect representation of the software, according to Maudet et al. (2017). Furthermore, such tools do not help to validate the design implementation during the development process continuously.

In a perfect world, designers fully participate in the implementation process to support the developers with ad hoc design decisions. However, this may not be feasible in reality due to various factors. For instance, manually checking a test version of the software for design changes is a time-consuming task. Furthermore, designers may not have the required technical knowledge to build and run the software regularly. While developers are aware of software changes through version control and conduct regular code reviews, such practices usually do not involve designers. Hence, designers may not be aware of new user interface changes unless they regularly assess the software. This situation is both inefficient and may risk that potential deviations from the intended design, or other issues, remain unnoticed.

1.2 Purpose of the study

The purpose of this study is to explore and evaluate how a novel software artifact could support the collaboration between designers and developers during the UI implementation process. The goal is to facilitate regular communication and thus catch design errors as early as possible. This goal is accomplished by first mapping the problems that lead to design errors, which then guides the creation of an artifact

that can be assessed.

1.3 Research questions

The following two research questions were devised based on the identified problems. Furthermore, questions were formulated to support the Design Science Research methodology in use, which is described in detail later.

RQ 1: What problems exist with the collaboration between designers and developers during the implementation of a UI?

RQ 2: What are potential solutions to mitigate problems that occur in the collaboration between designers and developers during the implementation of a UI? To what extent are the problems addressed by the solutions?

1.4 Disposition

This chapter introduced the study. In the next chapter, chapter 2, relevant theory is presented, both related to collaboration practices as well as the technical aspects that are key to understanding the proposed solution. Additionally, related work is presented to understand how this study relates to other similar initiatives. In chapter 3, the details of the DSR approach are presented. The output of the method is both results, from the data collection, and a software artifact. The final artifact is presented in chapter 4. The data that guided the design of the solution is presented afterward, in chapter 5. Findings and their implications are discussed in chapter 6, and finally, in chapter 7, conclusions are drawn.

2
Background

This chapter consists of three primary categories, each with a different purpose. Firstly related work is introduced to explain how this study relates and differs from existing initative. Secondly, we present studies that relate to the collaboration between designers and developers. This aims to describe the current practices and identify which problems are known to occur. The final part is a technical background that presents literature related to continuous pipelines, automated (GUI) testing and image comparison. This is relevant to understand the technical aspects of the artifact.

2.1 Related work

There are already various studies discussing designer and developer collaboration, its pitfalls, and suggestions of how these difficulties can be mitigated. Though the solutions vary in terms of characteristics, most of the relevant work we have identified can be classified into two categories; process-related practices and tool-related practices, both aiming at improving designer-developer collaboration but with different means.

One process-related approach is to follow a set of guidelines, such as presented by Friberg et al. (2017). These guidelines were formulated with the aim to help cross-functional teams work better and more efficiently together. Furthermore, additional guidelines and best practices exist. These are presented in detail in the next section as they are relevant to understand the current practices and problems within developer and designer collaboration. Such guidelines might indeed help improve the collaboration and thus reduce the occurrence of errors. However, this study does not compete with such initiatives. Instead, we aim to catch errors as early as possible. Furthermore, in order for the guidelines to be as effective as possible, the team is required to embrace and follow them. Additionally, the team might be required to change their process accordingly, which may not always be easy.

A vast amount of tools attempts to bridge the gap between the design and the UI implementation. Two common tooling approaches are either to make the design specification more well-defined or to enable the designer to produce the final UI implementation directly.

An example of a tool that aims to improve the design specification is Enact, as presented by Maudet et al. (2019). The tool specifically aims to facilitate the collaboration regarding touch-based interactions through gestures, which were found to be challenging to communicate with a developer. The purpose of the tool is to reduce the occurrence of design errors during the implementation by more thoroughly communicating the intended design, for instance, through interactive examples. While this tool may reduce the occurrence of errors, only touch-based interactions are in focus. Hence, other types of errors may still occur.

Similarly, Leiva et al. (2018) presents Montage, a tool to help designers capture video prototypes to communicate interactions more easily. In general, tools that aim to reduce the occurrence of errors differ from the solution presented in this study on a conceptual level; our solution help catching errors as early as possible, rather than attempting to avoid them entirely.

Model-driven engineering of user interfaces is an entirely different approach to UI design, in which the designer defines models that are used to generate the implemented UI directly. As explained by Vanderdonckt (2008), there are several known benefits, such as more easily being able to adapt to new requirements. However, the approach is also met with some major criticism, such as a high threshold for designers to learn and strict limitations on the resulting user interfaces (ibid.). Akiki et al. (2014) describes that a recent generation of model-driven UI development has the notable benefit of being adaptive and context-aware, in the sense that the UI can automatically adapt to suit both the user's needs and the platform. On a similar note, Rivero et al. (2014) presents mockup-driven UI development, which is a hybrid approach where design mockups are used to derive requirements for the development of model-driven user interfaces.

While model-driven engineering is an active field of research, it is not widely adopted in the industry. Akiki et al. (2014) notes that various prototypes and approaches have been presented in literature and that it might be time for a joint-venture with industry. While this approach might provide promising benefits and could reduce manual design implementation work, traditional UI development continues to dominate in industry, and at Intunio, for the time being. Hence, model-driven approaches are not considered in this study.

2.2 About Intunio

As noted, this study was conducted in collaboration with Intunio. This section describes the company, its services, as well as its software engineering process. This is relevant in order to understand the context of which the problems were identified, and solutions were proposed.

Intunio is a Gothenburg based software engineering consultancy specialized in the design and development of user experiences. Examples of clients are Ericsson, Electrolux, and Scania. The choice of technology depends on the needs of the client,

but web technology is most commonly used. Nine industry veterans from The Astonishing Tribe (TAT) and BlackBerry founded the company, and today it consists of a total of 16 employees. Out of the 16 employees, 13 work as either designer or developer.

Most of the client work is performed in-house in the form of projects. These projects often have a well-defined deliverable, such as a design prototype or a functional software solution. Intunio generally adheres to an agile software engineering methodology, but as the projects vary in scope and needs, they adapt the process accordingly. Some smaller projects may consist of just one designer and one developer and often has only a single final delivery. Such projects typically use a less formal process, as the designer and developer can collaborate tightly. Larger projects may consist of a small group of people and can last for multiple sprints with several deliveries over a more extended period. These projects often use a more rigorous process which includes standard agile practices such as daily stand-ups and sprint demonstrations.

In addition to in-house projects, Intunio employees sometimes work on-site as contractors, embedded in the client's software engineering team, and established software engineering processes. Hence, these employees are exposed to the client's process, which can vary significantly between different clients.

2.3 Designer-developer collaboration

While designers have become an increasingly important part of the development process (DeAmicis 2019), cross-disciplinary collaboration comes with challenges on its own. Cockton et al. (2016, p.2-3) notes that designers and developers traditionally follow different processes. Most commonly, modern-day designers adhere to a user-centered design (UCD) approach, which focuses on building empathy with the end-user in order to design solutions that meet their needs iteratively. Many UCD activities involve contact with the end-user, for instance, to understand the problem or validate whether a proposed design is satisfactory.

Integrating UCD into software engineering processes can be problematic. When the first attempts were made to integrate UCD, traditional waterfall methodology dominated, as explained by Cockton et al. (ibid., p.2-3). While the user research and user testing activities of UCD fit well into the waterfall process, the iterative design practice did not. In contrast, the iterative design practice fit perfectly with an agile software engineering process, but some activities, such as user research and user testing are given less room, as less planning is done up front.

As agile methodologies dominate in the industry today, this section focuses on how designers and developers collaborate within such a process.

2.3.1 Current practices

Delivering a new software feature requires collaboration from both the designer and the developer. The designer needs to design the UI beforehand, in order for the developer to implement it. Hence, the developer depends on the designer to implement the feature fully and vice versa. As the implementation is dependent on the design work, problems related to scheduling may arise, as found during the early attempts from 2007 to integrate design into the scrum development process at PayPal (Budwig et al. 2009). The company found that designers had to work long hours in order to stay ahead of the development team, as they started working on the same sprint at the same time. As a solution, Silva et al. (2013) suggests that the designer should be at least one agile sprint ahead of the development team. However, the authors also make clear that the designer must be available to developers in order to clarify and resolve issues regarding the design. This is in line with the adjusted process at PayPal from 2008 (Budwig et al. 2009), in which the authors note that the design team stays one or two sprints ahead, but also has time scheduled to support the development teams with the designs that are currently being implemented. While letting designers stay ahead of the development team seems to be a common approach, the designers and developers studied by Ferreira, Noble, et al. (2007) stated that they had noticed a positive benefit when collaborating more tightly throughout the iterations. Similarly, in a questionnaire by Jones et al. (2016), developers rated *access to designers* as the main factor for successful agile software development.

Friberg et al. (2017) studied the designer-developer collaboration practices at five software companies in Gothenburg, including Intunio. The authors found that the team members were physically seated either by profession or team, with different benefits to each approach. Team-based seating had advantages such as improving communication between designers and developers. Some preferred to be seated by profession in order to communicate more easily around technical problems. Similarly, a team observed by Ferreira, Sharp, et al. (2011) was seated by profession, and the authors found that the developers would physically move around the building to talk to the designers, only when they needed to discuss issues with the design work. The designers were never found to initiate contact with the developers, apart from delivering the design artifact.

2.3.2 Design hand-off

Designers and developers often perform the majority of their work in isolation (Maudet et al. 2017). A common practice is that the designer produces a design artifact before the implementation starts (Friberg et al. 2017), possibly a sprint ahead, as mentioned in the process by PayPal (Budwig et al. 2009). The finished design artifact is then handed over to the developer, who then proceeds to implement the UI. This procedure is called a design hand-off (Maudet et al. 2017). The study by Friberg et al. (2017) highlights that discussions seldom took place between the two disciplines during the preceding design phase. Furthermore, they also found that the designers were not integrated with the development process, and as a result, the disciplines lacked knowledge about each other's work. The authors note that

developers receive the design artifact first when the design is already set, which is something some of the interviewed developers found problematic.

The design artifacts can be seen as a blueprint that describes various aspects of the final version of the UI. Different types of design artifacts exist (Maudet et al. 2017), ranging from static mockups to more sophisticated interactive prototypes that also demonstrate interactive behavior, such as navigation between pages and animations. The static mockups appear as a screenshot of the desired UI, often created in vector format using interface design tools such as Sketch and Figma. Maudet et al. (ibid.) notes that a vast amount of prototyping tools have attempted to bridge the gap between design artifact and implementation; however, the tools do not yet produce production-grade software. As a result, a wide variety of tools and methods are used by the interviewed designers, in order to communicate various aspects of the design to the developer. However, visual design is often prioritized, according to a survey done by Myers et al. (2008).

Different teams have different routines for the hand-off procedure. The hand-off sometimes consists of a meeting, but not always. Friberg et al. (2017) found that delivery took place either via chat, email, or meeting, depending on the organization. Ferreira, Sharp, et al. (2011) note that the delivery occurred by email in their observed organization. Additional tools such as Zeplin (Zeplin 2020), which is used at Intunio, help facilitate the hand-off by turning the static mockup into a detailed specification, for instance, including sizes and color codes, and sometimes even code snippets, to help the developer be more precise and efficient.

2.3.3 Design breakdowns

Maudet et al. (2017) attempts to map the common problems faced during the implementation phase, based on 16 thorough interviews with both developers and designers. The authors define blocking issues that are found during the implementation phase as *design breakdowns* and divided into three categories;

Missing information: the designer has not fully specified the looks or interactions of certain parts, e.g., how a button should look while being hovered.

Edge cases: the designer has not considered extreme or problematic situations, for instance, empty states or very long usernames.

Technical constraints: the designer has not considered the technical limitations, leading to extra development time or complexity, e.g., having to implement an entirely custom UI component despite a slightly similar ready-made UI component being available in the system.

From a case study, the authors found that involving the developer in the design process did reduce the occurrence of these blocking issues in the implementation

phase. However, the issues would still occur nevertheless, for instance, due to misinterpretation based on the vocabulary mismatches between developers and designers.

Ferreira, Sharp, et al. (2011) found that the development team discussed the design artifact internally when they first retrieved it to identify potential issues beforehand. The type of issues mentioned was in line with the categories as defined by (Maudet et al. 2017), such as technical constraints. The feedback was then either sent by email or discussed in a meeting with the designers. The authors also noted that the developers were reluctant to improvise their own design solutions in the case of blocking issues such as missing edge cases. As an example, the designer might have designed the UI for a list of five elements, but not specified how the UI should look if there are no elements at all. The developers were reluctant to improvise their own solutions to such situations as they have had to rework their improvised solutions in the past once the designers noticed, leading to wasted effort.

Some designers manually create correction documents, such as annotated screenshots, during the implementation phase to communicate issues with the implementation to the developer (ibid.). Similarly, Ferreira, Sharp, et al. (2011) found that the developers either emailed or met with the designers for the same reason. The developers interviewed by Friberg et al. (2017) often gave feedback on the design during implementation, which took place in the form of a dialogue.

2.3.4 Feedback and review procedures

Friberg et al. (ibid.) found that the feedback practices varied between organizations and individuals. For instance, one developer always asked for feedback on the implemented design, before considering it as done. Generally, the authors note that, in most cases, feedback was given only when asked for. One developer said that they only gave feedback during their sprint retrospective, but thought that more regular feedback would be desirable. The authors also found that issues that were found after the release was hard to fix. This indicates that feedback during the implementation phase is important to avoid such situations.

Written communication was commonly used, mostly when the communication was not time sensitive, in order not to physically disturb the recipient. All organizations observed by Friberg et al. (ibid.) used some kind of chat tool internally. Chat was used for various purposes, including quick questions, feedback, and sharing files. The participants did however, feel that messages get lost in the history, which was bad for documentation. Chat communication would sometimes become messy and hard to follow, but the participants believed this to be solvable by using threaded communication more often. The most common chat solution was Slack, which is also used at Intunio.

In a workshop by Friberg et al. (ibid.) regarding how a desirable designer and developer workflow might look like, one group of participants said that they found

project documentation to be a waste of time and rather preferred it to be created naturally by recording the evolution of a project. They suggested this to be done by regular photographs and screenshots to make people see the design easily.

2.4 Continuous pipelines

Software development projects change over time, and hence the code base is often maintained through version control systems, such as Git. As developers make changes on their local machines, their individual codebases start to diverge from the one on the main branch. Continuous integration (CI) is the practice of developers merging and building their local changes on the main branch frequently, preferably many times a day, to avoid the previously mentioned problems.

Continuous delivery (CD) and deployment (CD) are extensions of continuous integration, extending the practice of merging and building often by also including testing and deployment of the software. The continuous pipeline is executed when new changes are pushed, and the new code has to pass all stages in the pipeline in order to be published. The pipeline is fully customizable and can include tests on all levels of hierarchy.

2.5 Headless browsers

A headless browser is a GUI-less browser that can be programmed to perform interactions, such as navigating to an URL, clicking a button or taking a screenshot. A notable example is Selenium, which is a web browser automation suite often used for automated testing of user interfaces (Selenium 2020a). Selenium supports any browser that implements the WebDriver specification by W3C (*WebDriver* 2018) and provides a unified API to control any such browser (Selenium 2020b).

Another notable example is Puppeteer by Google. The documentation states that the tool does not aim to replace Selenium but rather focuses provides additional features by focusing only on Google Chrome. Selenium is noted to focus on cross-browser testing, while Puppeteer bundles a single, compatible version of Google Chrome, which has the benefit of requiring zero setup (Google 2020).

2.6 GUI testing

Software can be tested on various levels of abstraction, such as unit and integration testing (Alsayed et al. 2017). As the GUI resides on a high level of abstraction, special tools are required to run automated functional tests, where the user interaction is simulated. From a design perspective, the valid state is manually tested through review procedures due to its subjective nature.

2.6.1 Manual reviews

Through reviews, both formal and informal, the software deliverable is manually tested by a team member or stakeholder. This procedure is conducted at the end of the development cycle, in order to verify that the software does not have any defects (Alsayed et al. 2017). For instance, the Scrum process consists of sprint reviews that take place at the end of each sprint, where the result is presented to the teammates and stakeholders (Schwaber et al. 2017).

One particularly thorough formal review method is inspection, as described by Wiegers (1995). Inspection consists of multiple activities apart from just identifying defects in an artifact, including rework and follow-up of the identified defects. Wiegers (ibid.) further notes that inspection helps catch errors early, which is much cheaper to correct early, rather than later, in the process. Furthermore, Alsayed et al. (2017) claims that the inspection method is more suitable for catching design errors compared to functional testing techniques.

2.6.2 Automated functional tests

Automated GUI testing tools help test that the GUI objects control the application as intended when interacted with. The interactions are either defined programmatically or by record and playback solutions (Meszaros 2003). As classified by Alégroth (2013), GUI testing can be divided into three generations. Each generation uses a different strategy for controlling the interaction, with different pros and cons (Alégroth et al. 2015).

1st generation. Interactions are defined by screen coordinates. While the approach works for any screen-based software, it would break if the GUI object is repositioned, which makes the test fragile.

2nd generation. The tests simulate interaction directly with the underlying GUI components. This approach is more resilient to visual changes, e.g., the position or look of a button may change, but the button is still identified as the same object as before. A downside is that this method is strongly tied to the GUI library and that interactions are only simulated in the GUI library layer. If the software is web-based, this is done with headless browsers, as mentioned in 2.5.

3rd generation. The coordinates of the GUI objects are identified through image recognition in order to interact with them. This is also defined as Visual GUI testing (VGT). This approach enables interaction with software without integrating with the GUI library and is less prone to the movement of GUI objects, but visual changes may still break the tests. For instance, Sikuli is a tool that lets the programmer define the tests using screenshots of the components, which are then used to identify the elements on the screen (Yeh et al. 2009).

2.6.3 Automated visual tests

As opposed to the above testing solutions that test software functionality, visual regression testing is a category of testing tools that identify changes in the GUI. For instance, the tool Wraith (BBC News n.d.) can capture a screenshot of a web page, which is then compared to a previously taken screenshot to identify visual differences.

2.7 Image comparison

In order to compute the difference between two images, the respective pixels need to be compared. While calculating the difference between two color values is trivial, various factors need to be considered for the result to be perceived correctly by the human eye. Firstly, as described by Vyšniauskas (2009), the human eye does not recognize the difference between all colors uniformly. Hence, the authors present a color comparison formula to mitigate the issue. Secondly, images often contain anti-aliased pixels, which is a result of a method of smoothing outlines that do not align with the pixel grid. Kotsarenko (2010) presents a way to detect anti-aliased pixels. An open-source library by Mapbox (2020) provides a simple solution to reliably calculate the difference between two images, by implementing ideas from both of these papers.

3
Methods

This study followed a Design Science Research (DSR) methodology. The central idea of DSR is to iteratively devise innovative artifacts in order to attempt to solve real problems. The methodology was chosen as it is known to be suitable for solving real problems within organizations while simultaneously contributing general findings to the research community (Dresch et al. 2014, p.v), which is in line with what this study was intended to accomplish. Furthermore, this study has been structured and conducted according to the guidelines by Knauss (2020) on applying DSR in the context of Master Thesis work in collaboration with the industry, particularly with regards to connecting the research questions to the iterations.

3.1 Design science research

Design science (DS) is the study of how to create artificial objects (artifacts) that satisfy desired requirements when embedded in an environment, such as an organization. Design science research (DSR) is the method of creating such general knowledge through the iterative process of designing artifacts that solve a specific problem in a specific organization (Vaishnavi et al. 2015, p.10-11).

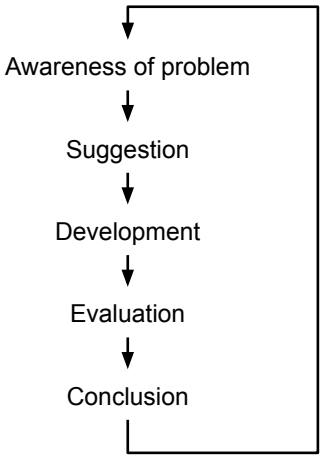


Figure 3.1: DSR process model

While various DSR process models have been suggested by different authors, there are many similarities between the models (Vaishnavi et al. 2015, p.18). This study follows the process defined by Vaishnavi et al. (ibid., p.14-17). As seen in figure 3.1 above, the process contains five steps that are performed in sequence, before starting over with another iteration.

Awareness of problem Awareness of relevant problems within the organization. This may be based on various sources, for instance industry experience or academic literature.

Suggestion Proposition of solutions that intend to solve the identified problems. This step is inherently a creative where a new solution is imagined, for instance, by introducing entirely new functionality or reconfiguring an existing solution in a novel way.

Development Creation of the proposed artifact. For a software artifact, this involves designing and developing the software.

Evaluation Evaluation of the artifact in relation to the problems it is intended to solve within the organization.

Conclusions. Clarification of learning achieved, conclusions, and generalization for a class of problems. While the artifact may solve a problem for a particular organization, the goal is also to generalize the knowledge as a research contribution.

3.2 Data collection methods

Several research methods were used as part of the DSR iterations in order to learn about the problems at the company as well as to evaluate the produced artifacts. The methods are formally defined in this section.

3.2.1 Literature review

While this study did not conduct a full systematic literature review, published literature was reviewed in order to identify the current practices and known problems related to the collaboration between designers and developers in the industry. Furthermore, topics related to automated testing a few specific technical topics were investigated to support the construction of the artifact. The relevant keywords included cross-functional teams, design inspections, code reviews, user interface implementation, integrating designers into agile development, automated GUI testing.

3.2.2 Focus group

A focus group was conducted early in this study in order to learn about the current collaboration practices and identify problems. Focus groups are recommended as initial orientation in a new field, as a wide variety of views and insights can be gathered with little effort (Longhurst 2003, p.106). The goal of a focus group is not to draw general conclusions, but to understand the perspective of a specific group (Krueger et al. 2015, p.63-67).

Krueger et al. (ibid., p.63-67) note that the purpose is key when deciding on the group composition. In general, the authors recommend homogeneity but with sufficient variation, in the sense that the participants have a relevant trait in common, but with potentially contrasting opinions to allow for interesting discussions.

This affected the selection of participants in the sense that every participant were either a designer or developer at the company. The participants were recruited specifically for their distinct roles, as they were assumed to bring a unique point of view. The purpose was to learn about their own experience as well as the perspective of their respective role.

Furthermore, Krueger et al. (ibid., p.67-68) state that focus groups normally contain ten to twelve participants, but that small focus groups of four to six participants are becoming increasingly popular due to practical reasons. A risk with fewer participants is that the variety of views is more limited compared to larger groups. In the case of this study, this was considered, but as the company is relatively small, it was deemed sufficient to recruit few but well-experienced people. That way, the participants have experience from a wide variety of projects and team constellations.

The focus group were conducted in-person and consisted of the following topics, each with some questions to initiate the discussion:

1. **Software development process**. What does your process look like? Do you use an agile process? Does it differ between projects?
2. **Code reviews**. Do you work with code reviews? How does it work? How often?
3. **Design hand-off**. How do you work with design hand-offs? How does it work? What is the design deliverable? Do you involve the programmers in the design process?
4. **Follow-up**. How do you follow up the design implementation? Do you have automated tests? How do you verify that a design is correctly implemented?
5. **Proposed solutions**. Would it help to track UI changes? What can go wrong during the implementation of a UI? What kind of aspects do you want to track? How can such aspects be tracked? Where should the information be made available?

3.2.3 Questionnaire

As explained by Gillham (2008, p.2-9), questionnaires are a cheap and common method of collecting viewpoints of many participants in a structured way. While questionnaires often consist of only closed questions, open-ended questions are suggested to lead to more discoveries, though with more effort required to analyze the answers. The author note that preceding a questionnaire with a semi-structured interview can lead to most answers already being discovered, and thus closed questions can be constructed based on the predicted answers.

A questionnaire was performed with developers and designers at Intunio. The questionnaire had two separate purposes. Firstly, it served to verify that the insight gained from the focus group applied to more people in the company, such as learning about commonly used communication tools and collaboration practices. Secondly, and more importantly, the questionnaire aimed to measure how the participants perceive the current designer-developer collaboration, as well as how often certain design errors occur.

The questionnaire in this study was preceded by a focus group, which allowed us to explore the topic before defining the questions. Still, the survey was semi-structured as a few open questions were believed to be useful to explore the topic further. The questions can be found in appendix A.

Some of the closed questions were defined using a Likert scale (Allen et al. 2007) with the intent of measuring the intensity in the answers regarding specific statements. Others were open to learning about their personal experience of the problems. While open answers, in general, require more effort to analyze, the number of answers was limited by the number of developers and designers at the company, and hence this was deemed acceptable.

Questions related to design errors were categorized according to the three design breakdowns by Maudet et al. (2017): missing information, edge cases, and technical constraints. These questions were part of those which used a Likert scale in order to capture how often the respondent perceives that such errors occur.
The response rate can be high if the participants are "captive" (Gillham 2008, p.9), which was the case in this study, as we personally requested the employees at Intunio to answer the questionnaire. Furthermore, the survey was intentionally non-anonymous to allow for individual comparison of how the participants believe that the artifact would affect their answers. This was used as part of the interviews, which are later described in 3.2.6.

3.2.4 Analysis of chat history

Each of Intunio's client projects has a chat channel consisting of only the relevant team members, including both designers and developers. Quantitative data was collected by extracting relevant conversations from the chat channels, from start to finish, of a specific number of past projects. A conversation was deemed relevant if

it related to the implementation of the UI, for instance regarding issues, questions or status updates.

The projects were selected based on the following criteria:

- The project consisted of both designers and developers.
- The project was built using web technology, as the artifact was only constructed to support web technology as a delimitation. While the chat history could be extracted no matter the technology, this was important as the same chat channels and projects would also be used for the simulation, as later described in 3.2.5.
- The company could give access without violating any confidentiality agreements.

Thematic analysis (Braun et al. 2012) was used to find common themes in the data gathered from observing historic chat conversations. The themes were derived from the data itself, based on the frequency of occurrence. This approach was suitable as the purpose was to explore the collaboration practices, rather than analyze them according to a predefined framework.

3.2.5 Simulation

As reviewed by Pries-Heje et al. (2008), there are various ways to evaluate an artifact as part of DSR. An artifact can be evaluated either before (ex-ante) or after (ex-post) the construction of the artifact. However, with design science research being iterative, the definition of ex-post relates to the artifact and not the final system, as noted by the author (ibid.).

Further considerations include whether to evaluate the artifact using a naturalistic method, such as observation, or an artificial method, such as experiments and simulation (Venable et al. 2012). The authors note that both approaches have their strengths and weaknesses. For instance, naturalistic evaluations risk misinterpretation due to the complex nature of reality, while artificial evaluations risk having the results not be applicable to reality. Furthermore, naturalistic evaluation, where the solution is tested in a real scenario, is mentioned as critical. However, it tends to be complex due to the many variables involved (Pries-Heje et al. 2008).

At the end of this study, a simulation was performed in order to assess the final artifact. The choice was made using the framework presented by (Venable et al. 2012). For instance, the choice of an artificial evaluation rather than a naturalistic one was partly due to practical reasons, as no suitable project was available at our host company at the time. Furthermore, the artifact is purely technical which is in line with the framework.

The simulation was performed as a replay of the project's UI implementation, by executing the artifact on every historic Git commit for a selection of suitable historical projects. This effectively shows how the artifact would behave and look given that

it would have been used at the time each commit was made by a developer. The past projects were part of the analyzed chat history (described in 3.2.4), meaning that for every chat conversation that related to a UI change, a commit where the change was introduced could be identified in the simulation data. For every such conversation in the chat history, an assessment was made of whether the artifact would have successfully captured a screenshot that could communicate the same context and spark a similar conversation. If so, the case was deemed supported in the sense that it would be sufficient for designers and developers to identify the problem in the simulated output of the artifact. The result of this was a percentage metric that states how many of the chat messages that initiated the design-related discussion, the artifact would already cover in an automated fashion.

Furthermore, to determine if the artifact could help the designers and developers to catch errors earlier than was previously identified manually, the timestamps were noted. This included both the time of the Git commit (i.e., when the UI change was introduced) and the chat conversation that related to the specific UI change. The difference between these two timestamps was used to determine how long it took for the problematic UI change to be manually identified and discussed in the chat. If the artifact would display the same issue directly at the time of the commit, we conclude that the feedback loop could be shortened if the artifact was used.

3.2.6 Interviews

Semi-structured interviews are known to be useful for obtaining qualitative data, such as opinions, from a limited number of people, as explained by Longhurst (2003, p.108-112). The authors state that these interviews share many similarities with focus groups, in the sense that the intention is not to obtain data from a representative sample of a larger population, but rather focus on people that have a specific relation to the topic. Hence, the participant selection is noted to be vital.

During this study, such interviews were conducted with the purpose of assessing the final artifact from the perspective of the target audience. The interviews were conducted using video calls. The artifact was demonstrated using simulated data from a project that the interviewee had participated in. The artifact was presented alongside an excerpt from a historic chat conversation that the interviewee participated in. The chat conversation regarded UI implementation issues in that particular project. This was done in order to give the interviewee a realistic example of how the artifact would function.

Participants were selected based on the following criteria:

- The person represented a distinct role among our interviewees (designer, developer, or a mix of both). This was important as the topic relates to the collaboration between the roles, and thus the different roles could assess the artifact from the perspective of their specific role.
- The interviewee has plenty of work experience. Similar to the focus group described above, vast experience is desirable to mitigate some of the risk with

interviewing only a small number of people, as their opinions would be based on multiple projects.
- The interviewee was present in the chat history of at least one of the three projects that were observed and simulated. This enabled us to demonstrate the artifact a project the interviewee would be familiar with.

The following questions were asked as part of the interview:

1. How do you think the artifact would have affected your previous answer to the following question: How often do you communicate, outside of process-related meetings (e.g., Scrum), with the opposite role (designer, if you work as a developer or vice versa) during a project?
2. How do you think the artifact would have affected your previous answer to the following question: To what extent are you aware of the implementation progress of your co-workers during a project?
3. How do you think the artifact would have affected your previous answer to the following question: If you're a designer, how confident are you that the design of a feature will be correctly implemented by a developer, with your current level of communication and feedback? If you're a developer, how confident are you that you implement the design of a feature correctly (from the designer's perspective), with your current level of communication and feedback?
4. Apart from what has already been answered, how do you think that the artifact would affect you and your collaboration?
5. What potential improvements do you see?
6. Do you have anything else to add?

Questions 1, 2, and 3 were based on the questionnaire in order to compare to the baseline. The interviewee was presented with his or her answer to the questionnaire question and was asked whether the artifact would be believed to have affected the answer. Furthermore, a few open questions were asked in order not to limit the answers. The interviews were analyzed using thematic analysis (Braun et al. 2012) to summarize and categorize the meaning of the answers.

3.2.7 Demonstration

In order to obtain feedback regarding the artifact, demonstrations were held twice as part of this study, one after the first iteration and one after the second. During the demonstrations, the current version of the artifact was showcased to a number of people at once, by navigating the artifact UI and orally explaining its functionality. The participants were asked to discuss a set of open-ended questions regarding the proposed solution, functionality, and UI of the artifact. Furthermore, potential flaws were discussed, and suggestions of how the flaws could be mitigated were given.

All of Intunio's employees were invited to attend the first demonstration, and hence a large number of participants with different roles participated, including business stakeholders. For the second iteration, an explicit selection of participants was made as the demonstration took place through a video call. The purpose was to ensure

that at least one designer and at least one developer participated.

3.3 DSR iterations

Three iterations were performed during the study, each of which included all of the five steps described. Each iteration contributed to answering both research questions to some degree, as the understanding of the problems and potential solutions gradually increased during the process. However, each iteration did have one primary research question in mind, which it aimed to contribute substantially to. Table 3.1 provides an overview of each phase of the three iterations. The primary research questions are noted next to the iteration number.

	Iteration 1 (RQ1)	**Iteration 2 (RQ2)**	**Iteration 3 (RQ2)**
Awareness of problem	Literature review Focus group Questionnaire	Analysis of chat history Feedback from demonstration	Feedback from demonstration
Suggestion	Facilitate regular communication regarding UI	Better track and communicate UI changes	Adapt to support specific projects
Development	Technical proof of concept	Functional artifact designed and implemented	Artifact adapted to support specific projects
Evaluation	Demonstration	Demonstration	Simulation Interviews
Conclusion	Aspects that need to be considered	Technical requirements to support specific projects	Assessment of usefulness

Table 3.1: Overview of DSR iterations

3.3.1 Awareness of problem

In order to gain an understanding of the problems (RQ1), the first iteration started out by a literature review, which was followed by a focus group and a questionnaire. As described in 2.3, there turned out to be no consensus in the industry on how designers should be integrated into agile software development processes.

Due to the lack of consensus, it was deemed important to assess the particular workflow and collaboration practices of the host company. Therefore, a small focus group and a questionnaire were carried out. The focus group had a duration of one hour and consisted of three senior employees, one being a designer, one being a programmer, and one being a mix of both. Based on the insights from the focus group, a questionnaire was constructed and sent to all of the company's developers and designers. The questionnaire was sent to a total of 13 employees who work as either designer or developer. Out of the 13 employees, nine answers were obtained.

Based on insights from the first iteration, a decision was made to analyze conversations from chat channels that were associated with past projects. This was performed during the second iteration using the method described in 3.2.4, in order to further learn about the current collaboration practices. The company provided

access to the chat history of three completed projects to map out the current collaboration practices further and identify problems. The projects were all built with web technology using React as the underlying framework. The chat channels consisted of a total of 14 unique employees, and a total of 39 messages were deemed relevant, as they initiated a conversation related to the user interface.

In both the second and third iteration, insights were obtained in the form of feedback, based on the demonstration that occurred as part of the evaluation of the previous iteration (later described in 3.3.4).

3.3.2 Suggestion

During the first iteration, various problems were identified. Solutions to the few problems that were thought to be the most interesting were considered further, both internally and through discussions with the company. The most promising approach consisted of a software artifact that facilitates regular communication between designers and developers with regard to the UI implementation process.

The evaluation from the first iteration revealed that the general idea of the solution was thought to be promising, but that certain aspects needed to be carefully considered in order for such a solution to be convenient for the company and hence solve the problem properly. Based on these findings, various alternative solutions were assessed (RQ2). A new solution was suggested where the artifact would properly track when the UI is changed, and better communicate these changes to the designers and the developers in order to initiate frequent communication.

During the third iteration, the solution was adapted to support a few specific projects were the artifact could have been deployed to, based on insights from evaluating the artifact during the second iteration. While the idea behind the solution did not change, this required a few technical changes as well as configuring the artifact to support the desired workflow of each project.

3.3.3 Development

This section describes the process of the development, but does not describe the artifact itself. The final artifact is presented fully in chapter 4, and the results that lead to the creation of the artifact are later presented in chapter 5.

During the first iteration, initial proof of concept was developed in accordance with the suggested solution. The intention of the artifact was partly to assess the technical feasibility, but also to communicate the idea to stakeholders at the company, to evaluate the solution.

Based on the findings from the first iteration and the revised suggested solution, a more sophisticated and functional version of the solution was designed and implemented. The new software artifact continued to use the underlying technology

as developed in the first iteration as a proof of concept, but was developed to be sufficiently functional to be tested in real projects.

After evaluating the artifact as part of the second iteration, certain technical requirements were identified to deploy the solution to a specific set of projects. Therefore, the third iteration served to implement changes to the software artifact according to the adapted suggested solution that met these requirements.

3.3.4 Evaluation

At the end of the first iteration, an evaluation was performed by demonstrating the artifact to 11 participants, both developers and designers. Our initial understanding of the problem was explained, and an initial artifact was presented, serving as a proof of concept of the suggested solution. The participants were invited to give feedback and assess if the artifact was on the right track to solve the given problem.

Similarly, a demonstration was performed at the end of the second iteration. The session consisted of four participants of both developers and designers. The participants were again asked to give feedback, but also specifically asked to consider what the technical requirements would be in order to deploy the solution to the projects where the solution was thought to be relevant.

At the end of the third iteration, a more rigorous evaluation was performed in order to assess to what degree the artifact would solve the problem (RQ2). Usage of the final artifact was simulated for the same three projects that were used for the chat analysis. A total of 785 commits were processed. Apart from simulations, interviews were conducted with five employees at the company.

3.3.5 Conclusion

From the first iteration, it was concluded that the idea behind the solution was considered promising by the company stakeholders and that the idea ought to be extended and refined in another iteration, to address certain issues and construct a functional artifact that can be tested.

The second iteration revealed that the solution had successfully addressed certain issues, but needed further changes in order to support the company's specific projects. Hence, another iteration was needed.

During the third and final iteration, the artifact was adapted to support the said projects, to perform the evaluation as described in 3.3.4. This resulted in insights about its believed effect towards solving the problem, primarily based on the assessment of the company stakeholders. The evaluation revealed several additional problems, including some potential ideas that might mitigate them. Hence, further iterations are believed to be beneficial to continue to improve the solution in order to solve the problem better.

4

The Artifact - Screeny

This chapter presents both the practical and technical aspects of the final version of the artifact, named Screeny, that was constructed according to the proposed solution. The final artifact is a result of the three iterations that were guided by the efforts to understand the problem and evaluate the artifacts. The design choices are later motivated in 5.1, as the findings are presented.

The artifact enables the designers to keep track of the UI implementation progress in order to facilitate communication between the disciplines. When the artifact is used with a software project, screenshots of the UI are taken automatically when a developer commits any change to the project's Git repository. The artifact determines if any visual changes were made compared to the last version. When the artifact detects a change, a summary of the changes is sent to a Slack chat channel, where both the designers and developers are present. The visual changes can then be viewed in full detail through a web application, which is easily accessible from the chat message.

4.1 Architecture overview

The artifact primarily served to evaluate the proposed solution in order to determine if it provides any benefit to designers and developers as part of the UI implementation process. Hence, some common functionality, such as user authentication, has been left out as it was deemed unimportant to fulfill the purpose. Furthermore, technical decisions were specifically made to meet the requirements of Intunio. A consequence of this is that the artifact only supports web technology and requires Bitbucket Pipelines to be used for continuous integration tooling. However, the technical aspects of the solution could be modified to meet other requirements.

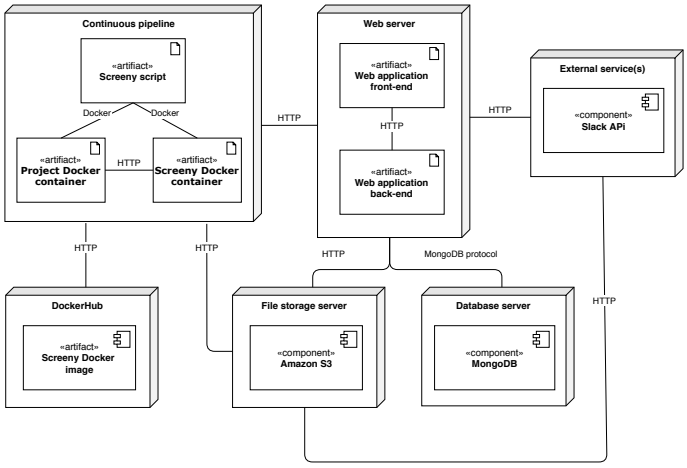


Figure 4.1: A deployment diagram of the Screeny artifact

A deployment diagram can be seen in figure 4.1. The prototype consists of three major parts; a script that runs as part of the continuous pipeline, a web application, and one or many external interfaces where the data is presented (e.g. Slack). The continuous pipeline interacts with DockerHub to download certain scripts, which will be described shortly. Additionally, the web application is supported by a database that is used for storing and querying structured data and a file storage service used for storing the screenshots. The continuous pipeline script uploads files directly to the file storage services, and the files can also be accessed by any external interfaces, in order to present the data.

4.1.1 Continuous pipeline script

The continuous pipeline script is denoted *Screeny script* and is responsible for taking screenshots of the software project. The script is configured to run whenever a commit is made to a Git repository. The repository also contains several files to set up and configure both the continuous pipeline, temporary project deployment, and screenshot automation.

4.1.1.1 Temporary project deployment

When continuous pipelines are executed, the project code is locally available in the temporary environment. A common task would be to run a test suite as part of the pipeline. Such a script would install any required software and dependencies, configure the project, and then run the test suite as a CLI program.

While similar steps are used for the Screeny script, there is a notable difference in that the project needs to be accessible from a web browser, and not just a test suite executed as a CLI program. This scenario is closer to a full deployment of the project, either in development or production. However, as the script acts as the web browser and runs in the same environment, local access is sufficient. This solution differs from continuous deployment in the sense that the project does not need to

be deployed publicly and to an external server. The project is only temporarily deployed, within the continuous pipeline environment, for the execution time of the script.

To solve this, Docker is used as part of the continuous pipeline. The project is required to have a "Dockerfile" that defines its software dependencies as well as how to configure and run the project's web server. If the project already uses Docker in either development or production, this requirement is already met. The continuous pipeline script launches a Docker container according to the project's Dockerfile, which then allows the project to be accessed through a web browser within the temporary environment.

Furthermore, the Screeny script in itself is temporarily downloaded to the continuous pipeline environment using Docker. The latest version of the Docker image downloaded from DockerHub, which is an official service to publish Docker images. This approach comes with a few benefits. Firstly, this ensures that the latest version of the Screeny script is run, as the image is downloaded from DockerHub first when the continuous pipeline is executed. Secondly, this avoids the need to store any Screeny script files in the Git repository of every software project, apart from the configuration that is specific to the project itself.

4.1.1.2 Screenshot automation

The primary purpose of the Screeny script is to obtain screenshots of the UI of the software project with which it is being run. To accomplish this, Google Chrome is controlled as a headless browser using Puppeteer (Google 2020). The browser is executed with a fixed screen resolution of 1440x900, which was deemed sufficient to capture ordinary desktop websites.

Screenshots are taken according to a list of page paths (e.g. */Team* representing *intunio.se/Team*), which is specified in the Screeny configuration file *screeny.js*. This configuration file is part of the project's Git repository. Apart from page paths, custom scripted interactions may also be specified in the configuration file by directly controlling the Puppeteer browser, for instance, to click a button to trigger a specific UI state. This approach adheres to the second generation of automated testing by directly interacting with the underlying GUI elements, as described in 2.6.2.

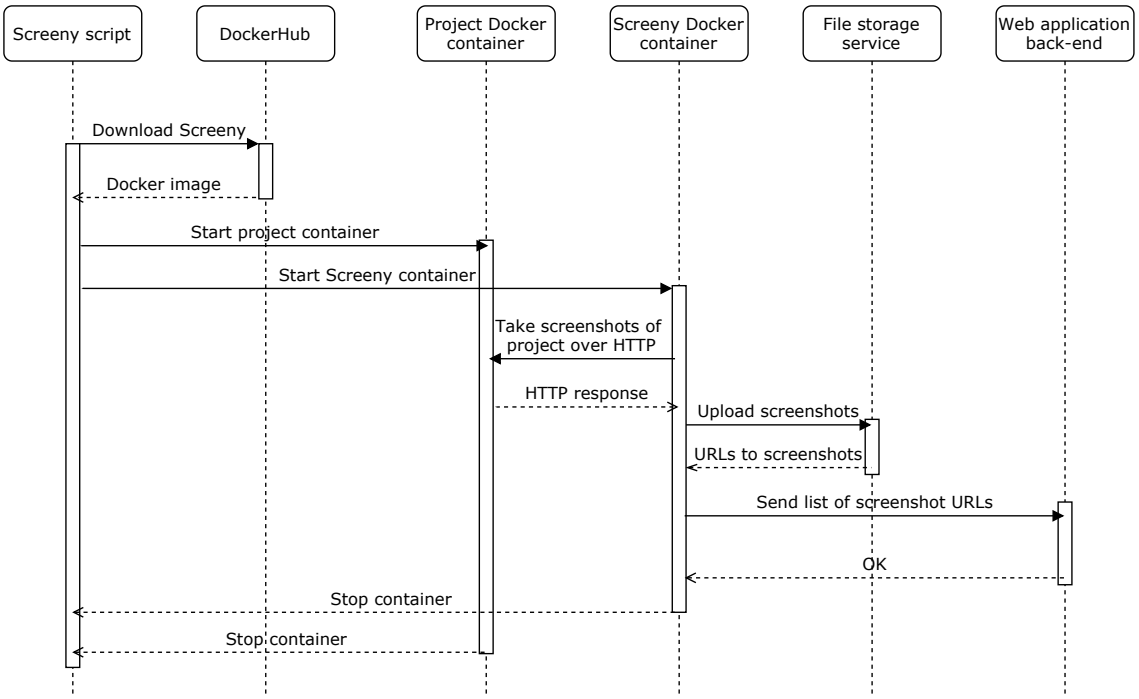


Figure 4.2: Sequence diagram of the continuous pipeline

When the continuous pipeline is run, the script starts both the project Docker container and the Screeny script Docker container, as illustrated in the sequence diagram in figure 4.2. The project container exposes a web server which is accessible via a specific hostname and port. For the Screeny script to have access to the temporarily deployed project, a Docker network is configured between the two containers. The Screeny script is then invoked to access the page paths and perform the interactions on the specified hostname (the Docker container name) and HTTP port (e.g. 8000), in order to take the desired screenshots. The screenshots captured by the headless browser are then uploaded directly to the file storage solution, as mentioned in 4.1, and the direct links to the uploaded images are obtained. Finally, the screenshot links are sent to the back-end using an HTTP request, together with metadata of the Git commit.

4.1.1.3 Configuration

The following few files need to be added to the project's Git repository to set up the artifact.

screeny.js The Screeny configuration file controls the screenshot automation using a list of page paths and custom interactions. The developers must keep this configuration file up to date to ensure that all desired pages are captured as the project evolves.

run-pipeline This file executes the continuous pipeline script and starts both the Screeny script and the project as part of a network, as described in 4.1.1.1.

bitbucket-pipelines.yml The Bitbucket Pipelines configuration file schedules the *run-pipeline* file to run when a commit is made, either to any branch or to a specific set of branches.

4.1.2 Web application

The central part of the artifact is a web application that both manages and presents all the collected data. The front-end consists of a simple interface that enables the user to explore the UI changes of a particular software project. The back-end primarily handles the incoming data from the continuous pipeline script.

4.1.2.1 Front-end

The web application consists of three pages; a list of projects, a list of commits within a project, and a detailed comparison of the UI changes that were made by a particular commit.

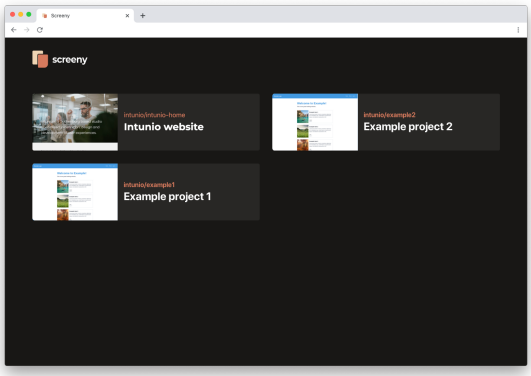


Figure 4.3: The list of projects

The list of projects can be seen in 4.3. This page provides a simple overview of all projects that are configured to be used with the artifact, with the most recent screenshot visible as a thumbnail image to allow for quick recognition.

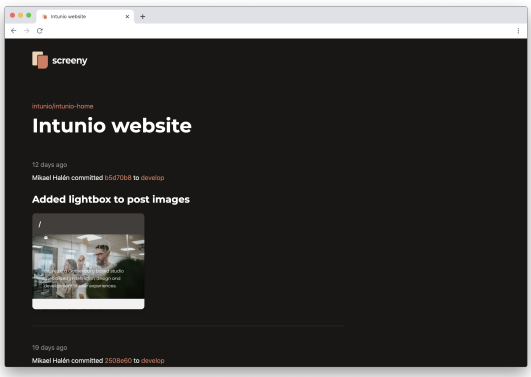


Figure 4.4: The list of commits for the "Intunio website" project

When navigating to a project, a chronological list of all the past commits are presented. This page is displayed in figure 4.4, however only the topmost commit is visible in the screenshot. The commits include relevant metadata such as the author, timestamp, and commit message. Most notably, if any UI was changed in the commit, screenshots of affected pages are presented as thumbnail images.

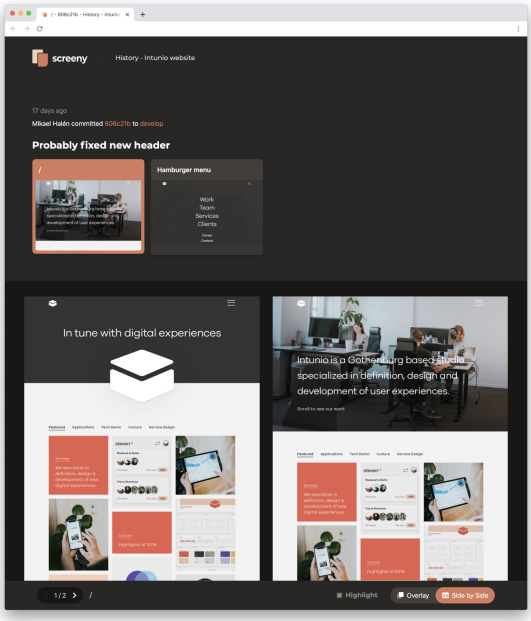


Figure 4.5: Detailed comparison of before and after the UI change (side by side)

A detailed comparison of the UI changes can be viewed by clicking one of the image thumbnails. The comparison has a few settings to help identify and understand the visual changes that were made. The *highlight* setting, if activated, places a red-tinted semi-transparent overlay on the areas that had visual changes. This feature is useful to help identify smaller changes, such as spacing and font sizes. Furthermore, two layout options are available. The *side by side* option shows the screenshots of the previous version (left side) and the new version (right side) next to each other. This can be seen in figure 4.5. If the highlight is enabled, it is only visible on top of the screenshot of the new version. The *overlay* option shows only a screenshot of the new version, with or without the highlight, though much bigger as it fills the entire screen.

4.1.2.2 Back-end

The back-end of the web application provides a few simple API endpoints for obtaining the data, which is used by the front-end interface. However, the most interesting yet complex aspect of the back-end is to retrieve, process and store the incoming screenshots and metadata from the continuous pipeline script.

For every commit, the back-end retrieves a list of screenshot URLs, as well as metadata about the commit. Each screenshot is uniquely identified by a key being either

the page path or a given name (in the case of a custom interaction). This value is denoted the *screenshot key*. A flowchart of the screenshot storage procedure can be seen in figure 4.6. First, the previous screenshot with the same key needs to be obtained to determine if the commit actually modified the captured UI. The two screenshots can then be compared to determine if they are identical or not. If the screenshots are identical, the screenshot with the particular key is not stored with the new commit, indicating that no UI change occurred.

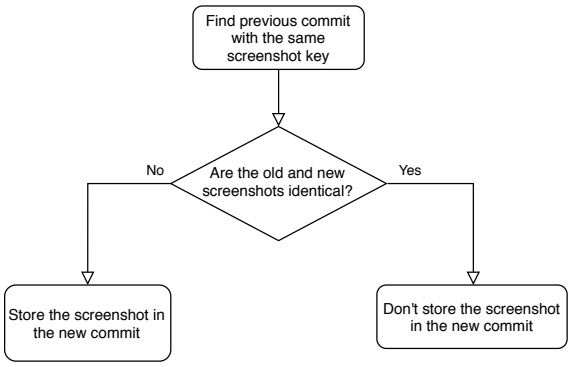


Figure 4.6: Flowchart of screenshot comparison procedure

Git commits are structured as a tree, with each commit being a node that typically has one parent node, and any number of children (in the case of branches). Hence, the commit that most recently changed the same part of the UI can be found by traversing the parents recursively and checking whether the commit has a screenshot with the same key, starting at the current commit. The algorithm is defined as pseudo-code in figure 4.7. If the traversal reaches the root, it can be concluded that the commit introduces a new page or state that was not previously seen.

```
getOldScreenshot(screenshotKey, commit)
    if the commit contains the screenshotKey
        oldScreenshot = get the screenshot with
         the same screenshotKey from the commit

        return oldScreenshot

    if the commit has a parent
        recursively run getOldScreenshot(screenshotKey, parent)
         to traverse the tree upwards
    else
        Root was reached: conclude that screenshotKey was not
         part of any previous commit
```

Figure 4.7: Pseudo-code of the algorithm used to find the most recent commit with the same screenshot key.

If a match is found, the images are compared using the pixel-by-pixel comparison library Pixelmatch (Mapbox 2020), which is based on the image comparison literature

described in 2.7. The library is used to detect if the two screenshots are identical, and if they are not, the library provides an image displaying the pixel differences in red. This image, which can be seen in figure 4.8 below, is used for the *Highlight* setting mentioned earlier.

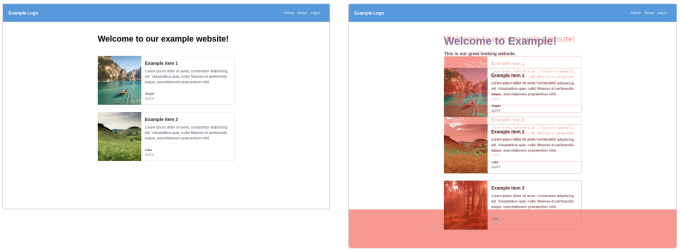


Figure 4.8: Visualization of the pixel difference between two screenshots

4.1.3 External interface (Slack)

Information about the UI changes can be sent to any number of external services. However, only Slack was added at the time of the third iteration.

By notifying a Slack chat channel of any UI changes, both designers and developers could easily follow the progress and discuss the changes. Technically, the back-end of the web application sends a message to a bot user within a predefined channel using the Slack API, whenever a UI change is detected.

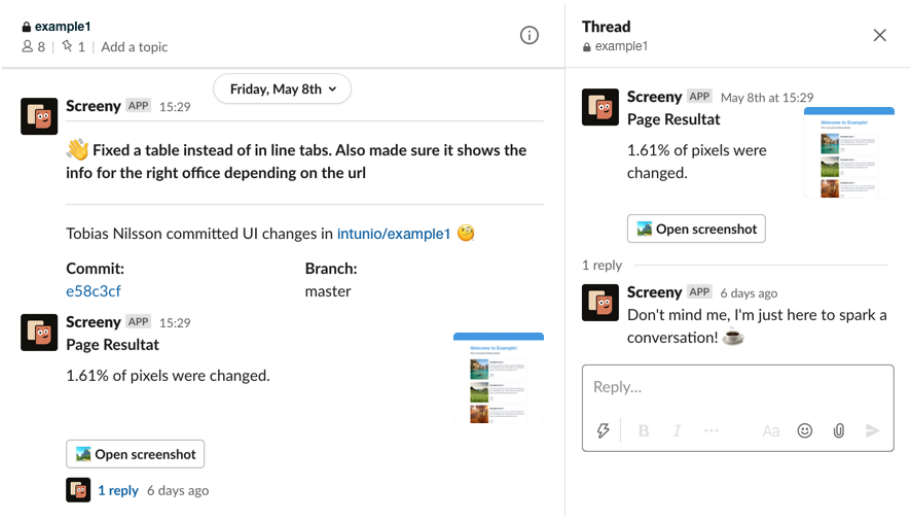


Figure 4.9: The Slack message with its corresponding thread conversation.

As seen in figure 4.9, the Slack message includes metadata, such as commit message and author, as well as a small thumbnail of the page. Furthermore, it presents a percentage indicating how many pixels were affected by the change. Most notably, the message includes a button which opens the web application with the detailed comparison of the particular UI change.

The bot also initiates a threaded conversation connected to the message, with the purpose of inviting the designers and developers to discuss the changes within the thread, to encapsulate the messages that relate to a particular screenshot and thus reduce the clutter in the channel itself.

4.2 Practical limitations

The artifact has several known limitations which have not been addressed due to time constraints.

- Screenshots are only taken using Google Chrome. As a result, cross-browser issues are not detected.
- Screenshots are only taken with a single, fixed resolution of 1440x900. Hence, alternative sizes, such as mobile and tablet, are not captured.
- The image comparison algorithm calculates the difference only from top to bottom, which is problematic when a new UI element is added above the existing UI. As the existing UI is pushed downwards by the new element, a difference is detected in almost every pixel.
- Only web-based software projects are supported, as the screenshots are taken by headless browsers. In theory, different platforms could be supported by changing the continuous pipeline script.
- Git merging is not properly handled. As a consequence, the artifact may not find a commit where the same page or state was previously changed, and hence falsely conclude that it was just introduced. In the case of a merge, a commit node has two or more parents, which was not handled properly as the issue was detected during the final evaluation.

5
Results

This chapter starts by presenting the data and findings from each of the data collection methods as part of the DSR iterations (see table 3.1 for an overview). Throughout this chapter, we derive six problems (denoted P.1 to P.6) based on a various data, both from the literature and from this study. These problems are defined by this study and serve as a motivation for the proposed solution. Finally the findings are summarized and grouped by the two research questions, including the list of the derived problems.

5.1 Findings per iteration

During each iteration, we identify problems, provide a suggestion, develop a solution, evaluate the solution, and finally conclude the findings. The section presents the main findings per iteration.

5.1.1 Iteration 1

The focus of the first iteration was to understand the underlying problems in designer and developer collaboration, which would guide the suggestion of a potential solution. To summarize, a literature review was done, a focus group was held, a questionnaire was sent out, the first artifact was created, and a demonstration was held, showcasing the artifact with the aim to gather feedback.

5.1.1.1 Literature review

The reviewed literature that relates the collaboration between designers and developers was previously presented in full, in 2.3 as part of the background chapter.

The work by Maudet et al. (2017) directly contributed to the understanding of the problems. The authors note that there is a gap between design and implementation, which the developer needs to bridge through interpretation. We defined this problem as a P.1. However, the most notable contribution by the authors is the definition of design breakdowns, which we defined as P.3. In related work by Maudet et al. (2019), it was made clear that interactions and animations are difficult to communicate, which led to the definition of P.4. Finally, Ferreira, Sharp, et al. (2011) found that developers create improved solutions on their own, which was defined as P.2.

5.1.1.2 Focus group

Insights from the focus group contributed to a general understanding of the collaboration between designers and developers at the company. Furthermore, all participants had vast work experience and provided personal anecdotes to showcase the problems they experience daily. The participants had experience both from projects at Intunio and at external clients. However, the participants did in some cases not specify from which project the experiences were based on. The group consisted of 3 employees, denoted persons A, B, and C. The employee roles were designer (A), developer (B), and one with expertise in both fields (C). A set of questions was beforehand to guide the discussion, which was described in detail in 3.2.2. In summary, the questions touched upon the current process and problems with regard to the UI implementation process. Lastly, potential solutions were discussed. The questions were informed by the literature related to agile processes and collaboration practices.

The employees at Intunio usually follow an agile process for most of their projects. However, the procedures vary from project to project, depending on the size and needs of the project. Examples of such procedures are meetings, code reviews, and sprint demos. Their projects are normally built by cross-functional teams, consisting of both designers and developers. The designers normally finish the design specification before handing it over to the developers, being one step ahead of the development process, in line with the literature described in 2.3. There was a mutual understanding between the designers and developers that a design specification does not fully cover all areas of the final software, mostly due to time constraints and efficiency. The developers were quite used to fill in the gaps where needed. This finding further confirms P.1 (improvisation).

Problems were often discovered late in development. Most notably, on one occasion, design errors in the implemented UI were noticed by the designer during the demonstration to a client. The participants suggested this stemmed from the challenge of keeping up with changes as the project is developed. To quote person A: *"In some cases, a designer designs a feature which then another person sends to a developer, who implements the feature which then a project leader accepts as finished, without acknowledgment from the original designer, even if the designer was not happy with the implementation"*. It became clear that this problem was rooted in the process of following up on the design implementation, which was either missing entirely or insufficient. Person C had tried to solve the problem himself by introducing a "gatekeeping" process where new changes in the implemented UI needed to be approved by him before being published. While this approach was time-consuming, it helped with catching errors before release. We defined the lack of awareness of the UI implementation progress as problem P.5. The proposed solution was greatly impacted by this insight in the sense that the core functionality help designers track the UI implementation.

Both person B and C manually took screenshots and videos and used them as a way of communicating status updates and feedback regularly, typically through Slack. Furthermore, the participants agreed that certain aspects of the user interface, such

as animations, are more difficult to convey than others, which confirms P.4 (animations). Both participants agreed that this was a slow and inefficient way of following up on implementation changes, which lead us to suggest a solution that would automatically post screenshots of UI changes to Slack, as a way of facilitating this communication.

Person A did not regularly follow the design implementation on his own due to technical difficulties with setting up the development environment locally. Occasionally, he tested the software at the developer's computer, however mostly when the developer invited him to do so. Person C explained that when working with web-based projects, there were cases where a staging server was set up, through which the designer could test the development version of the software. However, this was only the case in certain projects, even though the staging server was noted to be beneficial to the designer and helped assess the changes made during the implementation process.

Lastly, the focus group discussed what a potential solution would be. All of the employees wanted a simple way of following up changes made during development without adding too much overhead to their already established process. They mentioned that broadcasting changes through a project-specific Slack channel could be useful. However, person A argued that it would lead to even more daily notifications on Slack, which could be disturbing. Despite this, everyone preferred Slack rather than email. The argument of disturbance sparked a discussion of only notifying the team when changes had been made to specific git branches, for instance, the development branch. Furthermore, it was suggested that the solution should only trigger when it detects a UI change, i.e. not for every single commit.

5.1.1.3 Questionnaire

Answers to the questionnaire were received from nine of the Intunio employees. Five of the respondents are developers, three designers, and two a mix of both. One of the findings was that while all respondents use Slack and Zeplin, only the developers use Bitbucket. It was clear that communication was maintained regularly, with most respondents stating that they communicate either daily or a few times a sprint. Even though communication is practiced regularly, all designers answered that they were either neutral or quite unaware of the implementation progress during a project (P.5), which motivates the proposed solution of an automated way to track the UI implementation progress.

According to the questionnaire, most of the designers are confident or very confident that the design would be implemented correctly by the developer, with their current level of communication. However, two people felt that they were quite unconfident. When asked the same question but without the ability to communicate with the other role, all respondents but one downgraded their answers. Most notably, all designers downgraded their answers to either being unconfident or quite unconfident, which indicate a correlation between communication level and the confidence that the design would be implemented correctly.

In the following section of the questionnaire, the respondents were asked to rate how often they perceive that a certain design breakdown occurs. The categories of design breakdowns are defined as *Missing information*, *Edge cases* and *Technical constraints*, as previously explained in section 2.3.3. The questions were asked to both designers and developers. However, the errors are likely discovered by the developer during implementation. Hence, the designers were asked to consider how often they experienced that the developer asked for clarification or created improvised solutions where their design was not sufficient. The table below shows how the different roles responded to the different questions, divided by each *design breakdown*. The numbers indicate how often they perceive that the problems occur, one a scale from one to five (never to very often).

	1 (never)	**2**	**3**	**4**	**5** (very often)
Missing information	0	2	1	2	4
- Designer	0	1	1	1	0
- Developer	0	1	0	1	2
- Mix of both	0	0	0	0	2
Edge cases	0	0	2	3	4
- Designer	0	0	1	2	0
- Developer	0	0	1	1	2
- Mix of both	0	0	0	0	2
Technical constraints	1	1	2	4	1
- Designer	1	0	1	1	0
- Developer	0	1	0	3	0
- Mix of both	0	0	1	0	1

Figure 5.1: Heat map of the questionnaire results.

Looking at the *Missing information* row in the table 5.1, it is clear that the large majority of developers and multidisciplinary employees feel that the designer has not fully specified the design. The designers also felt that missing information was an issue, though to a lesser extent. The question regarding the *Edge cases* had similar results, though on the higher end of the spectrum, indicating that such errors happen quite often. The results to the final question regarding *Technical constraints* show that five of the participants perceive such errors to occur often or very often. At the same time, the rest were spread across the remaining choices. This data confirms problem P.3, concluding that all categories of design breakdowns are frequently occurring.

5.1.1.4 Demonstration

After the first iteration, the early artifact was demonstrated to 11 of the Intunio employees in order to gather feedback. Primarily, the intention was to validate the

idea behind the proposed solution and the functionality that was part of the artifact. In general, the participants liked the solution and the direction of which the artifact was heading. At the time, the artifact was only a proof of concept that took a screenshot of a web-based project whenever a Git commit was made. The screenshots were uploaded to a given Slack channel.

The participants thought that the Slack integration would fit their current workflow without adding too much overhead to their way of working. However, they were concerned about how often the Slack bot would post changes to the Slack channel. At the time being, the bot posted on every commit on any branch, making any other communication on the channel difficult due to the constant stream of bot messages filling up the screen. Furthermore, they also only wanted to be notified of visual changes in the project, not every commit. Furthermore, they wanted to more easily detect what part of the user interface changed compared to the previous version.

Based on these insights, the proposed solution was adapted to include both selection of which Git branches are relevant to be processed, and detection of image changes, in order not to notify the users of changes that are not relevant. Furthermore, the participants requested more clear visualizations of the UI changes. Though, as the customization options of Slack messages are limited, the solution was also extended to include a separate web application where the visual differences between screenshots could more clearly be highlighted. A direct link to the web application would be provided with each message in Slack to provide the users with convenient access.

5.1.2 Iteration 2

The main focus during iteration two was to further explore and develop a solution to the problems that had been identified previously.

5.1.2.1 Analysis of chat communication

Messages from three of Intunio's Slack project channels were analyzed using thematic analysis. The purpose was to strengthen the reliability of the results from the focus group and questionnaire. A total of 39 messages were considered as they related to UI changes. Themes were identified among these messages to detect common behaviors. The themes are presented in table 5.1, together with the frequency of occurrence. It was clear that both designers and developers were active in the channels, and communication was in many cases held daily between the group members.

Behavior	Frequency
Designer shares screenshot after testing the demo-version	8
Status update from developer with screenshot	6
Status update from developer	6
Feedback from other stakeholder	4
Status update from developer with video	4
Designers and developers discuss the technical implementation	3
Feedback from designer	2
Developer asks for design assets	2
Developer asks for design change	2
Designer shares video after testing the demo-version	2

Table 5.1: Results from the Slack channel analysis.

All three projects were automatically deployed either to production or a staging environment whenever the developer pushed a change to specific branches. As a result, the designers themselves were able to access the latest deployed version and assess the implemented UI, which was visible in the most frequent behavior. Another notable behavior was that developers regularly notified the chat channel, primarily the designers, that something had been deployed and could be assessed. With the notification, other stakeholders, such as business-related roles, also took the opportunity to share their opinions about the recent changes.

Based on these insights, the proposed solution was deemed to be beneficial in the sense that it would facilitate an existing behavior in a more automated fashion, reducing the need for manually sent screenshots and status updates. By reviewing the chat conversations, we found that the discussions could be relatively unstructured, which sparked the idea of encapsulating the discussions within threads concerning each screenshot to make discussions easier to follow.

5.1.2.2 Demonstration

The artifact was then presented to four of Intunio's employees at the end of the second iteration to obtain feedback and detect shortcomings. Additionally, the session aimed to investigate what blocking issues exist in order to run the artifact with some of their projects.
The participants liked the solution, just like during the first demonstration. Furthermore, they appreciated the progress that had been made with the artifact since the last time, indicating that it was on the right track. In particular, the newly added way of selecting a project and seeing all of its commits was appreciated. They noted that it would facilitate the process of tracking UI changes, especially historically, as they could scroll through a list of commits while seeing a thumbnail image of the UI changes for every commit. Furthermore, they liked the aesthetics of the web

application and its navigation. In general, the participants believed that the artifact would improve the way they collaborate, especially during the early phase of a project.

When asked about the blocking issues for running the artifact with their projects, the participants noted that the artifact, in its current state, would not work with the majority of the projects they had in mind, due to some of them being native mobile applications and some requiring dynamic interactions, such as logging in or clicking a button, in order to access all the pages of the project. The participants were also missing the ability to define custom screen resolutions of the screenshots as most of their projects are responsive and hence made for multiple devices, such as desktops and mobile phones.

The artifact would only fully support one of the projects in its current state. While native mobile applications are out of the scope of this study, the support for custom interactions was deemed reasonable to develop. Hence the artifact was further adapted to not only support taking screenshots through a list of page paths but also from scripted interactions, such as clicking a button (which was described in 4.1.1.2). Through this addition, the artifact could support two additional projects.

5.1.3 Iteration 3

During the last iteration, the focus was put on evaluating the final artifact.

5.1.3.1 Simulation

Simulation data were collected from three historical projects, denoted A, B, and C, consisting of a total of 785 commits. As seen in table 5.2, a total of 293 of the commits affected the UI, but only 18 chat conversations in the original projects' slack histories were related to UI changes. As noted, many UI changes were not discussed in Slack. Hence, the artifact could potentially have triggered conversations regarding UI changes that were previously not discussed. The simulation method is however, only able to compare the artifact's performance to the number of conversations that actually did take place.

Project	Total commits	UI changes	Chat conversations
A	60	44	2
B	200	70	1
C	525	179	15

Table 5.2: Simulation projects

The simulation resulted in a metric stating the portion of the 18 historic chat conversations that the artifact could potentially have initiated by a screenshot of the UI change. If so, the time difference between the UI change and the chat conversation

is calculated to determine how long it previously took to detect the error.

An example of a supported case can be seen in the two following figures below. Figure 5.3 illustrates an example of a historic Slack conversation by a designer who notes an error in the design implementation, which in this case was due to a text being wrong. The following picture, namely figure 5.2, shows the simulated output of the artifact for the same example commit. Though this time, it was not manually brought up by a designer but rather a follow-up of an automatically generated chat message. The same conclusion was drawn, i.e that the design implementation was faulty and needs correction.

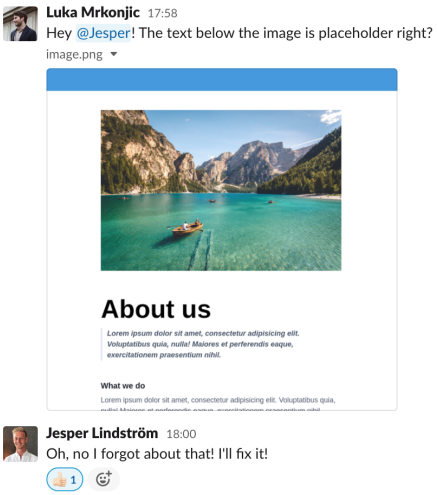


Figure 5.2: Slack discussion regarding a design error manually initiated by an employee

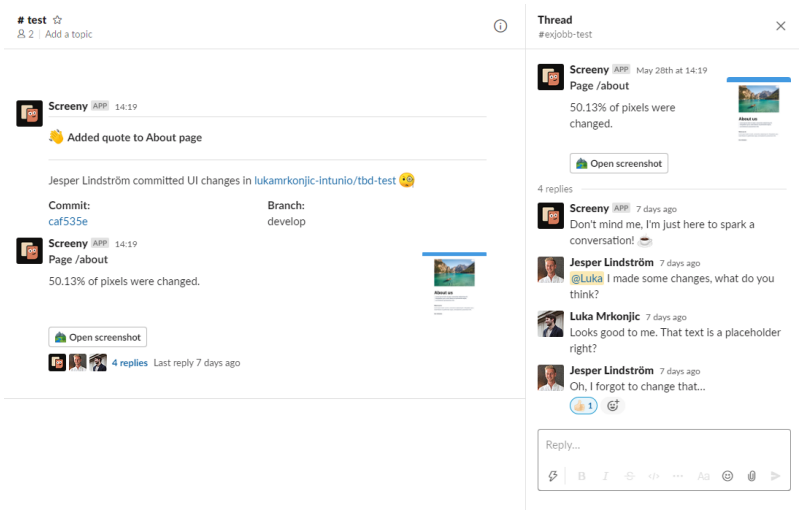


Figure 5.3: Slack discussion regarding a design error prompted by the Screeny Slack bot

Table 5.3 below presents the results of the simulation of the three projects. A row in the table is one conversation that relates to a UI change, based around one or multiple screenshots, in the project's Slack channel. The "supported" column states whether the artifact was able to take a screenshot of the UI change or not. Lastly, if the artifact supported the case, the time difference between the UI change and the chat conversation is noted in the last column. The difference is not calculated for unsupported cases as the commit can not be identified in the simulation data.

Case	Project	Supported	Difference (days)
1	A	No	-
2	A	No	-
3	B	Yes	39
4	C	Yes	52
5	C	Yes	0
6	C	No	-
7	C	Yes	42
8	C	No	-
9	C	No	-
10	C	No	-
11	C	Yes	1
12	C	No	-
13	C	No	-
14	C	Yes	0
15	C	Yes	57
16	C	Yes	57
17	C	Yes	0
18	C	No	-

Table 5.3: Simulation results

As seen in table 5.3, the artifact was able to capture screenshots that could support 9 out of 18 observed cases, i.e 50% of the time. Furthermore, with the supported cases, the difference in time between the UI change and the chat conversations was large in general, averaging at 28 days with a median of 39.

The reasons that the artifact was not able to support nine of the cases can be divided into the following categories. Three of the cases consisted of discussions that related to animations in the project, which the artifact did not support since it only captured static screenshots. Two of the cases concerned UI that required specific

interactions to access, for instance clicking a button to navigate to a specific UI state. While the artifact does support custom interactions, support for the specific cases need to be explicitly programmed, which was not the case during the simulation. Two of the cases were unsupported as they related to a responsive variant of the project, e.g. mobile UI, which is not supported by the artifact due to its fixed resolution. Finally, two of the cases were not supported due to structural changes in an externally hosted content management, which had changed since the commit was made. Hence, the case might have been captured if the artifact was run at the time of the actual commit.

Apart from the limitations of the artifact, screenshots from some commits could not be captured due to external software dependencies no longer being functional, which is a problem with the simulation approach, and not a limitation of the artifact. However, none of the chat conversations related to a commit that could not be captured, and hence the result is unaffected.

5.1.3.2 Interviews

Five employees from the company were interviewed to evaluate the artifact and to determine how they believe it would affect their collaboration. Additionally, the participants were asked to give general feedback on how to make the solution better. Out of the five interviewees, two were developers (denoted A and B), two were designers (C and D), and one was a mix between both (E).

A few questions which the participants previously answered in the questionnaire were asked once more. The participants' old answers were stated, and they were asked to consider if and how they believe the artifact would have affected their answers. All participants made it clear that the level of communication varies depending on the type of project. All participants thought that they would have communicated more had the artifact been present. Person D believed that the artifact would have been a good conversation starter, as they generally do not want to disturb one another and therefore are hesitant to initiate communication. We defined this finding as problem P.6. Furthermore, the perceived urgency of the communication would be reduced, in the sense that the designer could review the contents when available, rather than feeling the need to answer instantly.

When asked about their awareness of the UI implementation progress during the implementation process, all agreed that the artifact would have benefited their awareness. The designers thought the change in awareness would be improved majorly, though both developers and designers were believed to be positively impacted. Both designers agreed that being able to follow up on visual changes is beneficial in terms of general awareness during the project, as they sometimes lack the ability to run the project themselves.

Another question from the questionnaire regarded their confidence about the implementation being done correctly according to the design specification. Person B, a developer, explained that while trying to fill the gaps in lacking design specifications,

he misses the designer's original intentions and has to revisit his work in 8 out of 10 times. This confirmed problem P.2 regarding improvisation, though the interviewee did not seem reluctant to do so. He notes that the artifact would dramatically improve his way of working as it would lead to designers catching potential errors earlier. Person E notes that he usually reviews the implementation manually, many times during a project and that the artifact hence would have saved him both time and effort.

Lastly, the following are some concluding thoughts from each of the interviews:

- **Person A, developer:** *"The artifact would lower the barrier of communication between the roles and also save me time as a developer so that I do not have to, for example, send videos of my implementation progress."*

- **Person B, developer:** *"The feedback cycle from publishing a design specification, me implementing it and then sending it back to the designer is quite long. Using the artifact in our project would make it shorter and also lower the risk of design errors along the way."*

- **Person C, designer:** *"Using the artifact as means of verification is great as I can give feedback at my own pace, though, as I have not used the artifact in a real project, it is hard to imagine how effective it would be."*

- **Person D, designer:** *"The artifact would have helped me catch small, pixel imperfections in the UI implementation, which I mostly miss as they are usually hard to notice."*

- **Person E, a mix of both:** *"Initiating conversations and asking what others think can at times interrupt people's workflow. With the artifact in hand, I can definitely see how the process would have been much smoother."*

To summarize our results from the interviews, a thematic analysis was performed, as shown in table 5.4. The table is divided into three columns, *theme*, where common themes found in the interviews have been categorized, *No. of participants* noting how many of the interviewees brought up the selected theme during the interview and *theme description* where the meaning of the various themes are described.

Theme	No. of participants	Theme description
Facilitates communication	4 participants	The artifact would have facilitated the communication during a project.
Implementation awareness	4 participants	The artifact would have raised their implementation awareness during a project.
More confident	4 participants	The artifact would have made the participants more confident that the design is implemented correctly.
Initiates communication	4 participants	The interviewee mentions that the artifact would have made it easier to initiate communication during a project.
Catches errors	3 participants	The artifact would have increased the rate of catching visual errors in a project.
No custom resolutions	2 participants	The artifact lacks the ability to define custom resolution sizes.
No annotation support	2 participants	The artifact lacks the ability to annotate the screenshots while on the website.

Table 5.4: Thematic analysis of the interview results

Four out of five of the interviewees agreed that the artifact would have facilitated their communication and made it more consistent. While communication usually happens regularly, manually sending status updates of UI changes is easy to forget, as noted by one person. Four out of five also agreed that their implementation awareness would have been improved by using the artifact. Furthermore, four out of five believed the artifact would have made them more confident that a design is correctly implemented. While the developers themselves would not send status updates more often, the general implementation awareness in the team would likely be raised as the solution would automate the procedure. During the interviews, we learned that the participants were generally hesitant to initiate conversations in order not to disturb the other person (P.6). Four out of five participants noted that the artifact would reduce the bar to initiate conversations, and hence help catch visual errors, both earlier and at a higher rate.

While the interviewees generally had a positive attitude towards the artifact, a variety of potential improvements were also suggested. Two of the interviewees lacked the possibility of defining custom resolutions when using the artifact, which would have led to it missing screenshots of, e.g. mobile device screen sizes. Different participants had previously requested this during the demonstration of the second iteration (5.1.2.2), which indicates its importance. Two of the interviewees requested the ability to add annotations to specific areas of the screenshots to communicate more easily in connection to a specific part of the UI. As noted by one of the

interviewees, *"Annotating a particular area on the screenshot would have made it easier to communicate the problem to others, instead of trying to describe it in Slack."*.

5.2 RQ1 (Problems)

A variety of problems occur during the process of implementing user interfaces, with regards to the collaboration between designers and developers. The following problems have been identified and confirmed to occur, based on the literature presented and data collected at the company during this study.

- **P.1:** Design mockups are often incomplete, either intentionally due to resource limitations, or by mistake. As a result, developers need to fill in the gaps through interpretation.
- **P.2:** Developers sometimes create improvised solutions where the design is missing or incomplete. Developers regularly need to rework improvised solutions as they are occasionally deemed unsatisfactory by the designer.
- **P.3:** All categories of *design breakdowns* regularly occur during implementation due to insufficient design mockups, requiring rework or additions by the designer.
- **P.4:** Certain aspects of a design, such as animations, are difficult for designers to communicate to developers.

The following problems were identified in the data, yet only vaguely described in the literature, or not found at all.

- **P.5:** Designers are often not aware of the state of the UI implementation. As a result, issues with the design mockup or the implementation are discovered late in the process.
- **P.6:** Designers and developers are both generally hesitant to initiate communication, in order not to disturb the other person, despite that regular collaboration between the roles is known to reduce errors.

5.3 RQ2 (Solution Candidate)

The problems related to interpretation (P.1), improvisation (P.2), lack of awareness (P.5), and hesitation to initiate communication (P.6) were deemed the most relevant and promising for a potential solution to mitigate. This decision was made to reduce the scope of the study in order to meet the time constraints.

5.3.1 Proposed solution

As described in 5.1, potential solutions to these problems have evolved iteratively during the study. After three iterations, the proposed solution consists of a tool

whose primary purpose is to facilitate awareness and regular communication between the designers and developers throughout the implementation process. The details of our implementation of the final solution were described in chapter 4.

In summary, the tool regularly captures screenshots of the implemented user interface, whenever the developer publishes any UI changes to the code version control system. If the screenshots differ visually from the last version, the screenshots are automatically posted into a chat channel where both the designer and developer are present.

The automated screenshots allow the designers to keep track of the implementation, even without any contact with the developer (addressing P.5). Furthermore, this is thought to both help catch mistakes in the UI implementation (P.1) and to identify improvised solutions that may need the support of a designer (P.2). Furthermore, as the screenshot messages are sent automatically in a chat channel where both designers and developers can instantly communicate in connection to the message, the threshold of initiating contact with the other person is thought to be lower (P.6).

5.3.2 Evaluation

The interview and simulation data, obtained by evaluating the final artifact, indicates that the solution could mitigate the problems to some degree.

While errors related to interpretation (P.1) or improvisation (P.2) would still occur, four out of five interviewees thought that the solution would both facilitate communication and raise awareness of the implementation. Furthermore, three out of five interviewees noted that more errors would be caught using the solution. Thus, issues are believed to be identified earlier, and hence new issues that arise if the original issue is not mitigated might potentially be avoided entirely. The simulation data further confirmed this by showing that an average of 28 days passed between the introduction of an error and until it would be identified. In contrast, the solution would enable errors to be spotted when they are introduced, given that the solution is able to capture the error. The results show that the current state of the solution would capture half of the errors in our data set. However, the data set only consisted of a small number of conversations, yet the number of commits that caused UI changes were a few hundred (as seen in table 5.2). Hence, it is not clear whether the artifact could potentially have captured design errors in commits that were not discussed.

The interview data showed that the solution was believed to reduce the barrier of initiating contact (P.6) by four out five interviewees, as the communication would happen in connection to an automated message. It was suggested that an automated message was believed to be less urgent than if sent by a co-worker. However, during the focus group, one person noted that Slack notifications are disturbing, yet the participants of the interviews did not note that this would be problematic.

Finally, four of five interviewees believed the solution would help them stay aware of the implementation progress (P.5), which was believed to help designers verify the implementation. Furthermore, the solution was thought to make the participants more confident that the design is correctly implemented.

6
Discussion

The following chapter discusses the contributions of the report and lists the main threats to validity encountered during the study.

6.1 Contributions

In this study, we present Screeny, a solution to mitigate some of the problems that occur during the UI implementation process. Hence, we contribute an increased understanding of how such an artifact can be constructed to meet our specific goals. Most problems that this study identified were previously described in the literature, however, often based on case studies. Our findings contribute to these studies by confirming that the problems exist at yet another company, in particular a small-sized software consultancy. Through our confirmation, we incrementally increase the likeliness that these problems are general and widespread within software development.

The problems that relate to interpretation (P.1) and design breakdowns (P.3) were already well described by Maudet et al. (2017). Furthermore, in another study Maudet et al. (2019) describes that touch-based gestures and animations are difficult to communicate and hence prone to errors (P.4).

Improvisation (P.2) was described by Ferreira, Sharp, et al. (2011) in the sense that developers were reluctant to improvise solutions on their own. While we confirm the existence of improvisation, our findings do not show that the developers are reluctant to improvise, in contrast with the mentioned study, despite occasionally having to rework their solutions.

Notably, the following two problems identified were only briefly described in the literature.

We found that designers are not aware of the UI implementation progress (P.5). Similarly, Friberg et al. (2017) briefly noted that the designers were not integrated into the development process and hence lacked knowledge of each others' work. However, our findings also show that designers often are not only unaware of developers' daily work but are unaware of the state of the UI implementation in general. As a result of this, issues related to the UI implementation are found late in the process.

Finally, we learned that the roles are hesitant to initiate communication in order not to disturb one another (P.6). This problem was briefly touched upon by Ferreira, Sharp, et al. (2011) in the sense that the designers would never initiate contact. However, the designers in our study did initiate communication regularly, although both designers and developers were hesitant to do so.

6.2 Implications for practitioners

For the software industry, this study highlights the need for tight collaboration between designers and developers throughout the process in order to minimize errors. The proposed solution appears to provide some of the benefits of tight collaboration, without a change in process that would require the designer to participate fully in the implementation process. The implications are an efficient utilization of the designers' time, while at the same time, they are brought closer to the developers throughout the implementation process. This results in an increased ability to deliver high-quality software with fewer design errors.

6.3 Threats to Validity

Various factors risk the validity of the study and its generalizability. This section will go through the different validity threats based on the categorization by Runeson et al. (2012), i.e. internal, external, and construct validity.

6.3.1 Internal validity

As the study was carried out on-site, in tight collaboration with Intunio, the employees who participated in e.g. the evaluations were involved throughout the process. As a result, all employees who participated in the study already knew our intentions and showed interest, which might have lead to possible bias and a higher risk of skewed answers, in particular during the interviews where the artifact was to be critically assessed.

Finally, as no ongoing projects were suitable to test the artifact on at the time, none of the interviewees had actually used the artifact in a real project. Hence, the participants had to assess the artifact based on hypothetical answers to the interview questions, i.e how the artifact would have affected their collaboration.

6.3.2 External validity

Due to the qualitative nature and as the study was conducted at only one company, our findings are not generalizable. Furthermore, the context of the study was a small software consultancy company with less than 20 employees, which might not be representative of the software engineering sector in large. Additionally, the findings in this study are derived from qualitative data based on experiences and

opinions of a limited number of participants, rather than any statistical evidence.

Additionally, as the study only focused on a single company, the data collection was limited by the number of employees and projects. As a result, the data collection were generally restricted to a small population, further lowering the generalizability of the study.

Most employees at Intunio are experienced and well used to working with each other in different team compositions. Hence, problems that generally occur during the early phases of teamwork might have been overlooked. Furthermore, the employees might already be used to a process or solution that mitigates certain collaboration problems. These factors might impact the generalizability of our findings.

6.3.3 Construct validity

Various measures against low construct validity were taken. For instance, the risk of misunderstanding the interview questions were mitigated by demonstrating the artifact and thoroughly explaining the context. The interviews were also recorded and transcribed, further mitigating the risk of misinterpretation.

As the interviews were based on the interviewee's hypothetical, believed effects of the artifact, we can not ensure that the answers would be the same if the interviewees actually would have used the artifact as part of a real project. Furthermore, as the interviews and one of the two demonstrations were done remotely as video calls, we could not ensure that the interviewees fully understood the artifact's functionality.

The evaluation was affected by the lack of suitable ongoing projects at the company at the particular period in time. Hence, no naturalistic evaluation was possible, although it would have been desirable to obtain a stronger result. Instead, the described simulation method was performed using old projects and historic chat conversations, in conjunction with interviews where several participants provided a subjective assessment. However, the number of chat conversations were relatively small. Furthermore, we do not know if all the communication regarding the project took place in chat, as communication might also have occurred in person or through other platforms.

7

Conclusion

This study explored how the collaboration between designers and developers could be improved, specifically during the UI implementation process. A novel software artifact was constructed according to the proposed solution, as part of the DSR methodology, to catch errors as early as possible through the facilitation of regular communication between the roles.

Based on the literature presented and data collected at the company during this study, a set of six problems were identified to occur with regards to the collaboration during the implementation of user interfaces (RQ1). In order to narrow down the study due to time constraints only a handful of the problems were addressed by the proposed solution, namely P1, P2, P5 and P6.

- **P.1:** Developers fill in the gaps through interpretation.
- **P.2:** Developers improvise solutions.
- **P.3:** Design breakdowns occur regularly.
- **P.4:** Certain aspects are difficult to communicate, e.g. animations.
- **P.5:** Designers are often not aware the UI implementation.
- **P.6:** Designers and developers are hesitant to initiate communication.

A artifact was constructed to mitigate the selected problems by enabling designers to continuously track the UI implementation progress and thus identify errors early (RQ2). Furthermore, it also aimed to reduce the threshold of initiating communication to support more frequent collaboration.

The evaluation showed that the artifact is believed to have a positive effect on the problems it targeted. While errors are not prevented from happening, the artifact was thought to help catch errors earlier, as intended. For designers and developers, the artifact was believed to improve awareness, save time and bring an increased feeling of control. Furthermore, it was also believed to lower the barrier of initiating communication within the project. However, an evaluation has yet to be performed as part of a real project, as the findings were primarily based on the personal assessment of a limited number of participants.

7.1 Significance of the study

This study makes both academic and industrial contributions. To academia, this study brings an increased understanding of several problems already described in the literature, primarily by confirming that the problems occur at yet another company. Furthermore, we identified a few additional problems that were not clearly described in the literature.

In line with the DSR methodology, we also widen the understanding of how an artifact can be constructed to meet our specific goals, by proposing a potential solution that aims to mitigate a selection of the identified problems.

To industry in general, and Intunio in particular, this study contributes a concrete concept of a software artifact that could mitigate a selection of the identified problems to some extent.

7.2 Future work

The artifact has various practical limitations, as described in 4.2. As the evaluation indicated that mitigating these would cover a higher percentage of the errors, further iterations would be desirable. Additionally, the proposed solution only attempts to mitigate four of the six identified problems. The remaining two problems, P.3 (design breakdowns) and P.4 (animations), are left for future studies to address.

Two of the problems identified were not sufficiently described in the reviewed literature. Therefore we request future work to determine if these problems are local to our company, or if they apply in general.

As noted, we have learned from interviews that the proposed solution is believed to have a positive effect, yet we have not tested the solution in a real context. It is critical to perform a naturalistic evaluation as part of a real project to draw any conclusions about the effect of the solution.

Furthermore, this study was conducted together with just one small company. It is therefore desirable to evaluate the solution within different organizations, preferably of different sizes, to determine if the effect of the solution is applicable to other organizations and team constellations.

Bibliography

Akiki, Pierre et al. (2014). "Adaptive model-driven user interface development systems". In: *ACM Computing Surveys (CSUR)* 47.1, pp. 1–33.

Alégroth, Emil (2013). "On the industrial applicability of visual gui testing". PhD thesis. Chalmers University of Technology.

Alégroth, Emil et al. (2015). "Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study". In: *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, pp. 1–10.

Allen, I Elaine et al. (2007). "Likert scales and data analyses". In: *Quality progress* 40.7, pp. 64–65.

Alsayed, Alhuseen et al. (Feb. 2017). "IMPROVING SOFTWARE QUALITY MANAGEMENT: TESTING, REVIEW, INSPECTION AND WALKTHROUGH". In: *International Journal of Latest Research in Science and Technology* 6, pp. 2278–5299. DOI: 10.29111/ijlrst..

BBC News (n.d.). *Wraith*. Retrieved 2020-02-21. URL: http://bbc-news.github.io/wraith/.

Ben Nadel (2020). *The most common mistake engineers make during the designer-developer handoff*. Retrieved 2020-04-24. URL: https://www.invisionapp.com/inside-design/developer-engineer-handoff-mistakes/?fbclid=IwAR2FAePSy0bXZkS3eQMdQLKUys1LwxDsuI8gPw45bqXZJDLwZKfll_unVz8.

Braun, Virginia et al. (Jan. 2012). "Thematic analysis." In: pp. 57–71. ISBN: 978-1-4338-1003-9.

Budwig, Michael et al. (2009). "When user experience met agile: a case study". In: *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, pp. 3075–3084.

Cockton, Gilbert et al. (Jan. 2016). *Integrating User-Centred Design in Agile Development*. ISBN: 978-3-319-32163-9. DOI: 10.1007/978-3-319-32165-3.

DeAmicis, Carmel (2019). "The Decade of Design". In: IEEE. URL: https://www.figma.com/blog/the-rise-of-ux-ui-design-a-decade-in-reflection/.

Dresch, Aline et al. (Sept. 2014). *Design Science Research: A Method for Science and Technology Advancement*. ISBN: 978-3-319-07373-6. DOI: 10.1007/978-3-319-07374-3.

Ferreira, Jennifer, James Noble, et al. (Sept. 2007). "Agile Development Iterations and UI Design". In: pp. 50–58. ISBN: 0-7695-2872-4. DOI: 10.1109/AGILE.2007.8.

Ferreira, Jennifer, Helen Sharp, et al. (Aug. 2011). "User experience design and agile development: Managing cooperation through articulation work". In: *Softw., Pract. Exper.* 41, pp. 963–974. DOI: 10.1002/spe.1012.

Friberg, Julia et al. (2017). "Development of a Cooperation Framework for Cross-Functional Teams". MA thesis. Sweden: Chalmers University of Technology.

Gillham, Bill (2008). *Developing a Questionnaire.* Real World Research Ser. Bloomsbury Publishing Plc. ISBN: 9781441154866. URL: http://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=cat07472a&AN=clec.EBC1644312&site=eds-live&scope=site&custid=s3911979&authtype=sso&group=main&profile=eds.

Google (2020). *Puppeteer | Tools for Web Developers | Google Developers.* Retrieved 2020-02-21. URL: https://developers.google.com/web/tools/puppeteer.

Jones, Alexander et al. (Jan. 2016). "Collaboration Contrainsts for Designers and Developers in an Agile Environment". In: DOI: 10.14236/ewic/HCI2016.37.

Knauss, Eric (2020). *Constructive Master Thesis Work in Industry: Guidelines for Applying Design Science Research.*

Kotsarenko, Yuriy (2010). "Measuring perceived color difference using YIQ NTSC transmission color space in mobile applications". In: *Programación Matemática y Software.*

Krueger, Richard A. et al. (2015). *Focus groups : a practical guide for applied research.* Sage Publications. ISBN: 9781483365244. URL: https://www.sagepub.com/sites/default/files/upm-binaries/24056_Chapter4.pdf.

Leiva, Germán et al. (2018). "Montage: A Video Prototyping System to Reduce Re-Shooting and Increase Re-Usability". In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, pp. 675–682.

Longhurst, Robyn (2003). "Semi-structured interviews and focus groups". In: *Key methods in geography* 3, pp. 143–156.

Mapbox (Apr. 2020). *mapbox/pixelmatch.* URL: https://github.com/mapbox/pixelmatch.

Maudet, Nolwenn et al. (2017). "Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems". In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pp. 630–641.

– (2019). "Enact: Reducing designer–developer breakdowns when prototyping custom interactions". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 26.3, pp. 1–48.

Meszaros, Gerard (2003). "Agile regression testing using record & playback". In: *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 353–360.

Myers, Brad et al. (2008). "How Designers Design and Program Interactive Behaviors". In: *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing.* VLHCC '08. USA: IEEE Computer Society, pp. 177–184. ISBN: 9781424425280. DOI: 10.1109/VLHCC.2008.4639081. URL: https://doi.org/10.1109/VLHCC.2008.4639081.

Pries-Heje, Jan et al. (2008). "Strategies for Design Science Research Evaluation." In: *ECIS*, pp. 255–266.

Rivero, José Matías et al. (2014). "Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering". In: *Information and Software Technology* 56.6, pp. 670–687. ISSN: 0950-5849. DOI: https://doi.org/10.1016/

j.infsof.2014.01.011. URL: http://www.sciencedirect.com/science/article/pii/S0950584914000226.

Runeson, Per et al. (2012). *Case Study Research in Software Engineering*. John Wiley Sons Inc. ISBN: 9781118104354. URL: http://www.egov.ee/media/1267/case-study-research-in-software-engineering.pdf.

Schwaber, Ken et al. (2017). *The Scrum Guide*. Retrieved 2020-02-21. URL: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

Selenium (2020a). *Selenium automates browsers. That's it!* Retrieved 2020-02-21. URL: https://selenium.dev/.

– (2020b). *Understanding the components*. Retrieved 2020-03-30. URL: https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/.

Silva, Tiago Silva da et al. (2013). "Ten lessons learned from integrating interaction design and agile development". In: *2013 Agile Conference*. IEEE, pp. 42–49.

Vaishnavi, Vijay K. et al. (2015). *Design Science Research Methods and Patterns: Innovating Information and Communication Technology, 2nd Edition*. 2nd. USA: CRC Press, Inc. ISBN: 1498715257.

Vanderdonckt, Jean (Jan. 2008). "Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges". In: *Proceedings of Annual Romanian Conference on Human-Computer Interaction*.

Venable, John et al. (May 2012). "A Comprehensive Framework for Evaluation in Design Science Research". In: vol. 7286, pp. 423–438. DOI: 10.1007/978-3-642-29863-9_31.

Vyšniauskas, V (2009). "Anti-aliased Pixel and Intensity Slope Detector". In: *Elektronika ir Elektrotechnika* 95.7, pp. 107–110.

WebDriver (2018). Retrieved 2020-03-30. URL: https://www.w3.org/TR/webdriver1/.

Wiegers, Karl E (1995). "Improving quality through software inspections". In: *Software Development* 3.4, pp. 55–64.

Yeh, Tom et al. (2009). "Sikuli: using GUI screenshots for search and automation". In: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pp. 183–192.

Zeplin (2020). *Zeplin*. Retrieved 2020-03-24. URL: https://zeplin.io/.

A

Appendix 1 - questionnaire

Intunio - baseline questionnaire

*Obligatorisk

1. What is your name? *

2. Do you primarily identify yourself as a designer or developer? *

   Markera endast en oval.

   - Designer
   - Developer
   - Both designer & developer

3. What collaboration software do you normally keep up with during a project? *

   Markera alla som gäller.

   - Slack
   - BitBucket
   - Trello
   - Jira
   - Zeplin
   - Övrigt:

4. How often do you communicate, outside of process related meetings (e.g. Scrum), with the opposite role (designer, if you work as developer, or vice versa) during a project? *

   Markera endast en oval.

   - Never
   - Once per sprint
   - A few times per sprint
   - Daily

5. To what extent are you aware of the implementation progress of your co-workers during a project? *

   Markera endast en oval.

   - ◯ Unaware
   - ◯ Quite unaware
   - ◯ Neutral
   - ◯ Quite aware
   - ◯ Aware

6. If you're a designer, how confident are you that the design of a feature will be correctly implemented by a developer, with your current level of communication and feedback? If you're a developer, how confident are you that you implement the design of a feature correctly (from the designer's perspective), with your current level of communication and feedback? *

   Markera endast en oval.

   - ◯ Unconfident
   - ◯ Quite unconfident
   - ◯ Neutral
   - ◯ Quite confident
   - ◯ Confident

7. How would you answer the previous question again, without the possibility of communication and feedback during the implementation process? *

   Markera endast en oval.

   - ◯ Unconfident
   - ◯ Quite unconfident
   - ◯ Neutral
   - ◯ Quite confident
   - ◯ Confident

How often do the following situations occur during the implementation of a design? This question applies to both developers and designers. If you're a designer, how often have you experienced that the developer asked for clarification or made his/her own solutions where your design was not sufficient?

8. – Missing information: the designer has not fully specified the looks or interactions of certain parts, e.g. how a button should look while being hovered. *

Markera endast en oval.

	1	2	3	4	5	
Never	◯	◯	◯	◯	◯	Very often

9. – Edge cases: the designer have not considered extreme or problematic situations for instance empty states or very long usernames. *

Markera endast en oval.

	1	2	3	4	5	
Never	◯	◯	◯	◯	◯	Very often

10. – Technical constraints: the designer has not considered the technical limitations, leading to extra development time or complexity, e.g. having to implement an entirely custom UI component despite a slightly similar ready-made UI component being available in the system. *

Markera endast en oval.

	1	2	3	4	5	
Never	◯	◯	◯	◯	◯	Very often

11. Are there any other types of problematic situations that you have experienced during developer-designer collaboration?