EXAMENSARBETE INOM MEDIETEKNIK,
AVANCERAD NIVÅ, 30 HP
STOCKHOLM, *SVERIGE 2021*

Defining and Evaluating Design System Usability for Improved Developer Experience

ELLA WIRSTAD GUSTAFSSON

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Abstract

The recent growth of the software industry has drawn attention to strategies aimed at improving collaboration within the development team. Shown to frequently encounter dilemmas associated with cross-disciplinary work are User Experience designers and software developers, whose effective collaboration are essential for the success of the development process. *Design systems* have evolved as a response to the demand for efficient collaboration as a collection of reusable patterns and documentation used to establish a common language within software development teams. The popularity of using design systems within software development projects has grown rapidly in recent years, but little research has yet been conducted on its usability in development work.

The goal of this thesis is to study design systems from a developer perspective, and evaluate its ability to facilitate the collaboration within software development teams. By building upon previous work on developer experience, this study proposes a method for assessing design system usability from a developer perspective. In this study, participating software developers were tasked with implementing user interface design in code with the help of a design system, followed by interview questions based on the Cognitive Dimensions Framework. Thematic analysis was used to analyze the data. Results provide insights into design system usability and developer experience. Findings include aspects of design system usability in general, such as the importance of developers' autonomy and flexibility of the system, as well as implications from the tested method in particular, such as the relevance of context of implementation. The results implicated an inherent dilemma of approaching design system development through conventions and best practices rather than with a systems theory and systems thinking perspective.

Sammanfattning

Senaste tidens tillväxt inom mjukvaruutvecklingsindustrin har resulterat i ett ökat intresse för att effektivisera samarbetet inom utvecklingsteamet. Två yrkesgrupper som påvisats stöta på dilemman associerade med multidisciplinärt arbete är de som designar och utvecklar mjukvara, vars samarbete är avgörande för utvecklingsprocessens framgång. *Designsystem* har vuxit fram som ett svar på en ökad efterfrågan på effektivt samarbete som en samling återanvändbara mönster och dokumentation som syftar till att etablera en gemensam vokabulär och inom mjukvaruutvecklingsteam. Trots att användandet av designsystem inom mjukvaruutveckling har ökat under de senaste åren har inte mycket forskning gjorts på dess användbarhet i utvecklingsarbetet.

Denna uppsats syftar till att studera designsystem från ett utvecklarperspektiv, och utvärdera dess förmåga att facilitera samarbetet inom mjukvaruutvecklingsteam. Genom att ta avstamp i tidigare forskning kring utvecklarupplevelse föreslår denna studie en metod för att bedöma användbarheten och utvecklarupplevelsen av ett designsystem. Deltagande mjukvaruutvecklare fick i denna studie i uppgift att implementera gränssnittsdesign i kod med hjälp av ett designsystem, följt av intervjufrågor baserade på ramverket för Cognitive Dimensions. Tematisk analys användes för att analysera data. Resultaten ger inblick i användbarhet och designsystem i allmänhet, så som vikten av utvecklarens autonomi och systemets flexibilitet, samt implikationer från det testade design systemet i synnerhet, så som implementationskontextens relevans. Resultaten antyder att det finns svårigheter med att närma sig mjukvaruutveckling med designsystem genom konventioner och god praxis snarare än med ett systemteoretiskt perspektiv.

Defining and Evaluating Design System Usability for Improved Developer Experience

Ella Wirstad Gustafsson
KTH Royal Institute of Technology
Stockholm, Sweden
ellag@kth.se

ABSTRACT

The recent growth of the software industry has drawn attention to strategies aimed at improving collaboration within the development team. Shown to frequently encounter dilemmas associated with cross-disciplinary work are User Experience designers and software developers, whose effective collaboration are essential for the success of the development process. *Design systems* have evolved as a response to the demand for efficient collaboration as a collection of reusable patterns and documentation used to establish a common language within software development teams. The popularity of using design systems within software development projects has grown rapidly in recent years, but little research has yet been conducted on its usability in development work. The goal of this thesis is to study design systems from a developer perspective, and evaluate its ability to facilitate the collaboration within software development teams. By building upon previous work on developer experience, this study proposes a method for assessing design system usability from a developer perspective. In this study, participating software developers were tasked with implementing user interface design in code with the help of a design system, followed by interview questions based on the Cognitive Dimensions Framework. Thematic analysis was used to analyze the data. Results provide insights into design system usability and developer experience. Findings include aspects of design system usability in general, such as the importance of developers' autonomy and flexibility of the system, as well as implications from the tested method in particular, such as the relevance of context of implementation. The results implicated an inherent dilemma of approaching design system development through conventions and best practices rather than with a systems theory and systems thinking perspective.

Author Keywords

Developer Experience; Designer-Developer Collaboration; Software Development, Design Patterns, Design System

CCS Concepts

•**Human-centered computing** → *Usability testing;* •**Software and its engineering** → **Collaboration in software development;**

INTRODUCTION

Agile development practices have evolved to enable rapid and stable development of new software that can quickly adjust to novel technological advances. The integration of User Experience (UX) into agile software development has been a well-discussed topic within the software development industry and an active research topic within academia, and has proven to offer a multitude of advantages for the quality of the product, user experience and team satisfaction [17]. Although clearly rewarding when functional, recent research recognizes that such cross-disciplinary integration and work is highly complex and depends on equal trust and verbal communication within the team in order to exchange design knowledge between team members [4]. Despite being tasked with the common goal to build quality software, developers and designers approach software development from different perspectives, oftentimes leading to friction between them [18]. Previous studies exploring methods for bridging the gap between design and development have identified differences in vocabulary within the team, late developer involvement [27] and lack of a shared design vision [32] as a source to miscommunication. Moreover, Agile development, as described in the Agile Manifesto [5], focuses on addressing software engineering issues, which has shown to sometimes be at odds with the user-centered design tradition of UX designers [17]. As a result, multiple tools and practices have been developed and tested over the years to facilitate the task of communicating design, but redundancy, mismatches and missing information are issues collaborating designers and developers still manage in their everyday work [27].

In recent years, the concept of design systems has gained popularity among industry practitioners, as an attempt to gather helpful tools and practices into one communicative single source of truth [30]. A standardized definition of the term remains to be determined, but commonly entails a collection of guidelines and resources used within development teams to document information with the goal to facilitate the implementation of digital product design. The design system aims to provide a shared language for all stakeholders — such as managers, development team and marketing team — involved in the software development process. Moreover, the arrangement of user interface design into systematically organized components resembles software developers' way of structuring components of code, which implies an improved software development process as design practices approach developer techniques.

This study aims to examine software development practices in order to answer the following research questions:

1. *How can design systems be defined and developed to improve developer experience and usability?*
2. *How can design system usability be evaluated in terms of developer experience?*

Case settings

This thesis is conducted in collaboration with the software development company Intunio AB, for whom the results of this study will offer insights into how design practices affect their employees' internal communication and perceived working experience. Intunio's expertise includes working closely with software development and user interface design in order to create design systems and other digital products for their customers. For software development companies such as Intunio, and their customers, this thesis aims to provide guidelines and best practices from a developer perspective for working with design systems in the development processes.

BACKGROUND

This section presents previous research relating to systematic organization of design, developer experience and designer-developer collaboration.

Software development collaboration

Although tasked with a unified goal to build quality software, developers and designers approach software development from different perspectives, sometimes causing miscommunication and friction in their collaborative work [18]. Moreover, a systematic literature review showed that restricting factors of the integration of User Centered Design and Agile methods include lack of design documentation such as wikis, design patterns and tool support and a shared understanding of the design vision [32]. In a study evaluating practices for delivering and interpreting interaction design, differences in vocabulary within the team and late developer involvement were identified as causes of miscommunication [27]. Furthermore, according to Brhel et al., communication for agile and user-centered software development can be mediated through artifacts such as prototypes, user stories and wireframes which "should be used to document and communicate product and design concepts, and should be accessible to all involved stakeholders" [10, p. 22]. These types of artifacts are also called boundary objects, which describes objects with the ability to mediate information between actors from different cultural, social and scientific backgrounds to facilitate cooperation in diverse groups [34]. Moreover, developers do not always use shared artifacts such as personas and scenarios, as designers intend [7]. This indicates a need for shared information and artifacts to be accessible and usable for the developer in order to prevent lost information.

Systematic design in industry and academia

The practice of capturing design knowledge as a set of interconnected patterns was firstly adopted by Alexandrian patterns for architecture in the 1970's [3]. Within Human-Computer Interaction (HCI) research, the concept of formalizing a shared language of design patterns and practices in order to address recurring design problems has been covered by researchers since the end of the 20th century [35] [26]. Besides providing a lingua franca — a common language — for all stakeholders of a design project [14], pattern libraries also offer a way for design to approach reusable software and object-oriented systems [19].

Lately, industry practitioners have explored ways to approach software development practices further through the formation of *design systems*. A standard definition of the term *design system* has not been established in either academia or industry, where similar terms, e.g., *pattern library* and *style guide*, tend to be used interchangeably [22]. However, a standard definition of design system is "a set of connected patterns and shared practices, coherently organized to serve the purposes of a digital product" [21, p. 14]. In order to grasp the full range and potential of a design system, a clear distinction from similar terms is necessary. In the adopted standard definition, the *pattern library* which is a broad set of *design patterns* or reusable parts of the interface, the *style guide* containing styles such as iconography, colors and typography and component library which is a set of implemented design components, are all subsets of the design system. According to this definition, a design system should also describe shared practices i.e. "how we choose to create, capture, share and use those patterns" [21, p. 12]. Moreover, this definition shares several similarities with a commonplace definition of a system, described within systems thinking theory as "a relationship of parts that work together in an organized manner to accomplish a common purpose" [11]. The idea of using systems thinking to address fuzzy, human-centered issues, such as the ones emerging within software development, has been used within design theory as well as other disciplines long before the formation of design systems. This principle was originally formulated in 1973, as a way to deal with societal problems, described as *wicked* dilemmas, which should be approached and responded to intelligently rather than as problems to be solved [29]. Furthermore, the principle of using systems thinking to approach complex dilemmas has also been embraced by Systems Oriented Design (SOD), where modern systems thinking and systems theory in the context of design, constitutes an instrument for designers to address complex challenges [33].

Developer experience and usability

In order to optimize software development team performance, a comprehensive understanding of developers' perceived experience of working within various project environments is essential [16]. Influenced by the ISO standardized concept of UX [2], the concept of Developer Experience (DX) was introduced as a way to address the experience of software development [16]. In contrast to UX, DX addresses experience relating to the creation of software rather than the use of software [23]. In turn, UX is commonly considered an extension of the well-recognized and standardized concept of usability, although they are sometimes used indistinctly [31]. The usability ISO standard describes it as "the extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [1]. Similarly, in

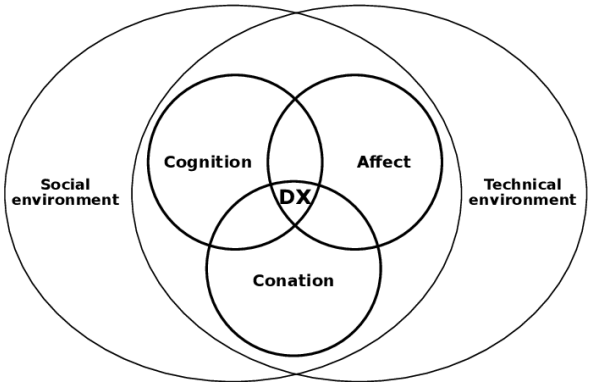

Figure 1. Developer Experience as described by Fagerholm [15]

the context of DX, research suggests evaluating the usability of developer tools operationally based on comparisons using the cognitive dimensions framework, as a way to approach the subjectivity of DX in programming tasks [6].

Design system as a developer tool

Design systems have evolved as a response to a demand for efficient collaboration and a common vocabulary within software development teams. Unlike other commonplace tools for communicating design, e.g. prototypes, personas and scenarios, design systems are not primarily 'owned' by designers, in the sense that they do not always require designers' expertise to deploy, administer, operate or interpret them [14]. Design systems can thus be considered a tool for all stakeholders accessing it, particularly user interface developers who are working closely with design. Evaluating the usability of a design system as a developer tool, from a DX perspective, could thus improve its ability to support developer practices and improve developer performance. Moreover, as described in Figure 1, DX includes various mental processes and environments forming through interactions between the developer, other people and "artifacts that a developer may encounter as part of their involvement in software development" [15, p. 55]. Based on this notion, various studies have been conducted to evaluate the DX of developer tools. For instance, intrinsic motivation and sense of flow have shown to be significant predictors of DX when evaluating the DX of an Integrated Development Environment (IDE) [24]. Furthermore, the cognitive dimensions framework have successfully been used to evaluate usability of an Application Programming Interface (API) and provide new insights into API design and usability from a developer perspective [28].

METHOD

This section presents the design and setup of the study conducted to explore methods for evaluating design system usability from a developer perspective.

Study design

The goal of the study was to examine design systems as a tool used by developers to interpret a communicated design vision, and explore methods to evaluate what aspects of a design system are associated with a positive developer experience. By adapting a method based on the framework of cognitive dimension [28] — formerly used to evaluate usability of an API — valuable views and aspects of design system usability were explored. The main part of this study consisted of a coding task, where 5 participating developers were tasked with implementing user interface design in code, with the assistance of a design system. Qualitative data connected to design systems was collected through a semi-structured interview based on four framework-adapted usability aspects; understandability, abstraction, reusability and learnability. Information concerning participants' working tasks and experiences of working with design systems was also collected prior to the coding task through semi-structured interviews.

Design system

As the purpose of this study was to explore methods to evaluate design system usability, the creation of the tested design system was not of main focus, but rather the methods of evaluation and aspects of design system usability in general. As such, the collaborating company Intunio provided basic components from their digital design system, *Kitten*, which were implemented in code for the sake of this study. For the rapid creation of a testable design system, a service for creating design systems, *zeroheight*, was used. The structure of the tested design system was largely shaped by the utilized service, influenced by Google's Material Design and based on Kholmatovas' definition of design systems. As shown in Figure 2, the Kitten Design System included resources such as logotypes, icons and color variables, a style guide declaring color schemes, typography and layout and design, implementation of modular UI components and application principles.

Adaptation of API usability evaluation method

The subject of evaluating usability of programming environments is multifaceted and inherently subjective due to individual programmers' habits and preferred ways of working. Researchers within similar areas have historically approached this dilemma by using the cognitive dimensions framework, describing the usability of notational systems [6]. In one empirical study, [28] this framework was used to evaluate the usability of an API. Similar to design systems, APIs provide access to reusable library components, and aim to support developers in their programming tasks by enabling modularity and abstraction of implementation details. In order to assess what characteristics of an API makes it easy or hard to use, the API usability study adapted Clarke's dimensions of API usability and the cognitive dimensions framework [12], and conducted a study where students and professional developers were recorded while solving a number of programming tasks using an API.

In order to evaluate the usability of a design system, this study adapted twelve interview questions based on four essential usability aspects from the cognitive dimensions framework [28]. The interview questions were adapted to conform with this study's goal to investigate design system usability. Rather than to prove usability, the interview questions aimed to discover the existence of usability issues in the design system, which is the recommended approach to the cognitive dimensions framework [13]. The purpose of the interview

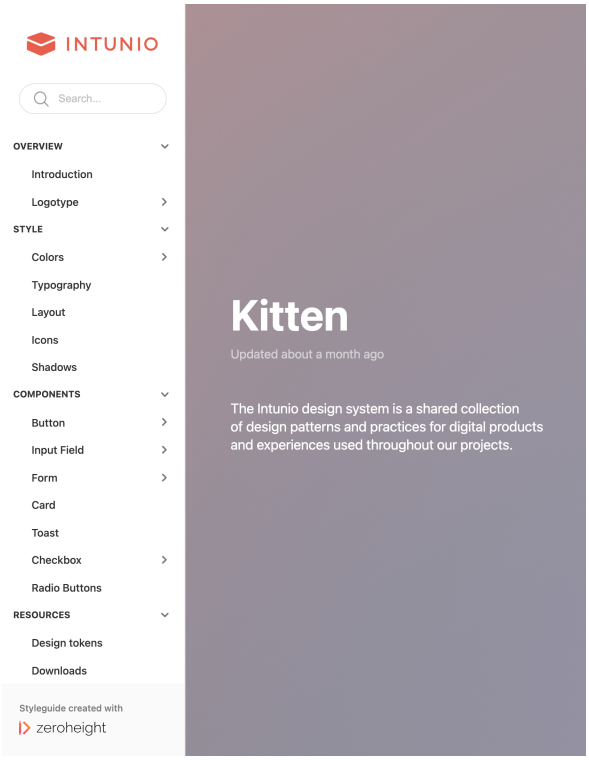

Figure 2. Detail view of the Kitten Design System structure

questions was to evaluate the design system usability against the following four usability aspects: understandability, abstraction, reusability and learnability. In the present study, *understandability* was adapted to address the effort required to understand the semantics of the design system's features based on names and documentation. *Abstraction* refers to whether the abstraction level of the design system aims to simplify without disabling the possibility of complex operations. *Reusability* examines whether the design system facilitates developers' capacity to implement design with concise, maintainable and reusable code. Lastly, *learnability* explores if the design system enables developers to learn how to use the design system easily and incrementally. For this study, the twelve adapted interview questions were, as follows, grouped by the usability aspect they target.

Questions regarding understandability:

1. Do you find that the design system maps to the design scenario in the way you expected?
2. Do you feel you had to keep track of information not represented by the design system to perform the task?
3. Does the code required to solve the tasks match your expectations?

Questions regarding abstraction:

4. Did you make any decisions or assumptions of your own, in order to implement the design scenario?
5. Could you rely on any conventions or previous experiences when using the design system to perform the task?
6. Do you feel the design system helped you perform complex subtasks?

Questions regarding reusability:

7. Does the amount of code required to implement the design seem about right, too much, or too little for you?
8. How easy was it to reuse code?
9. Do you feel you had to choose one way (out of many) to perform a subtask in the scenario?
10. Do you feel you would have to change much of the code to use the design system components in a different project?

Questions regarding learnability:

11. Once you had performed a few subtasks, was it easier to perform the remaining tasks?
12. Do you feel you had to learn many details specific to the design system to perform the task?

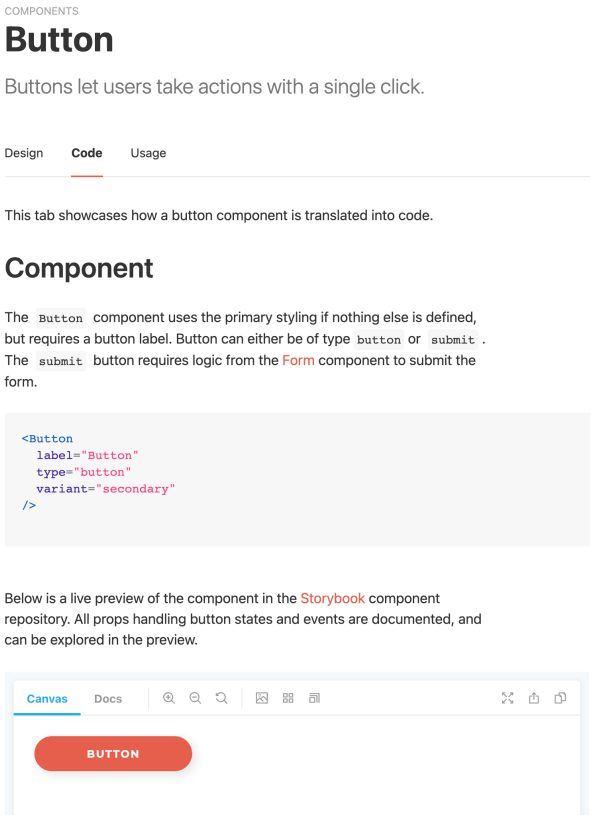

Figure 3. Code documentation of button component in the Kitten Design System

Study setup

In order to gain insight into design system usability from a developer perspective and investigate methods for evaluating design system usability, this study consisted of one initialising interview, one coding task and one finishing interview. Both interviews and the coding task were conducted and recorded using the video conference call application *Zoom*. Interviews were transcribed and data was collected from the transcriptions using a thematic content analysis.

Coding task

To explore the research questions, five developers were recruited and tasked with implementing design with the support of a design system. The design system provided participants with documentation describing how to use UI components and other assets needed to solve the task, as shown in Figure 3. A development environment was set up prior to testing in order to obtain consistency throughout the study, and keep participants focused on the main purpose of the study. The environment was set up using the command-line-tool *Create React App*, which sets up a pre-configured web application with the JavaScript framework *React*. The recruitment process had assured all developers were accustomed to the chosen technologies.

The *design scenario* i.e., a single-page design mockup to be implemented by the developers in the coding task was also developed. The goal of the design scenario was to create an authentic setting where the developers would use the design system as a tool to implement design. Participants were encouraged to use the design scenario as a guide to help them explore the design system rather than replicating every pixel of the design, giving them space for their own interpretation in order to stimulate participants' thoughts, reactions and reflections while performing the task. For the sake of maintaining consistency throughout the study, and to make sure all developers solved the same tasks, the design scenario was divided into 6 subtasks. If, at the end of the test, any of the developers had not completed one or several of the subtasks, they were performed together with the conductor of the study. The tasks, as represented in Figure 4 were:

1. Download the logotype asset from the design system and use it in the code.
2. Find design tokens for typography and apply the intended styling to the text.
3. Find and use the input field component correctly inside the form component.
4. Find and use the form component correctly to validate the input field and checkbox components.
5. Find and use the checkbox component correctly inside the form component.
6. Find and use the button component correctly to enable submission when validation succeeds.

Method for data collection

In order to learn more about participants' working tasks, views and experiences, interviews were conducted with participants

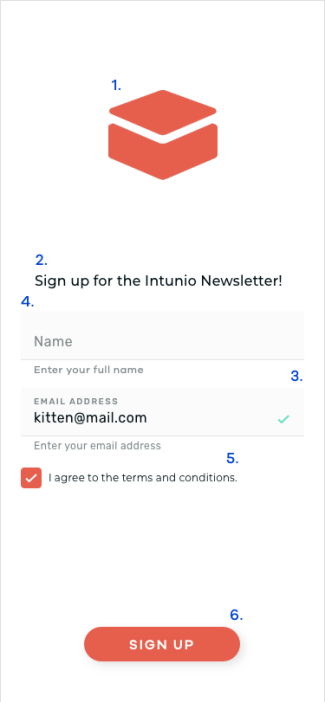

Figure 4. The design scenario divided into 6 subtasks

prior to introducing the coding task. As the purpose of the initial interviews was to explore participants' attitudes towards designer-developer collaboration and design systems, they were conducted in a semi-structured manner, enabling the conductor to abbreviate and add questions to the original protocol. Secondly, the coding task was initialized by introducing participants to the purpose and layout of the task. After being asked to implement design with the help of the design system, participants were handed the design system, a pre-configured development environment and a mockup describing the user interface design they were tasked to implement. Moreover, participants were asked to follow the thinking-aloud protocol in order to record their experience fulfilling the task. The thinking-aloud protocol is frequently applied within usability testing to collect data from users through verbal reports while using a system in order to explore their mental processes as they develop [8]. Recordings from the thinking-aloud were not analyzed, but aimed rather at encouraging and initiating discussions during the second interviews. Lastly, based on the twelve adapted interview questions, interviews were conducted in order to identify usability issues of the design system. For this interview, the semi-structured format was chosen to enable additional abbreviations and explanations to the original set of questions.

Methods for data analysis

The adapted API usability study used usability tokens to analyze data from participants' performances. However, since the present study was of a more explorative nature, a thematic analysis [9] was performed to identify valuable themes related to design systems and DX as formulated in the first research question. The thematic content analysis offered a more flexible

Participants

Number	*Position*
1	Developer and designer
2	Developer with design background
3	Front-end developer
4	Front-end/app developer
5	Front-end developer

Table 1. Participant number and position

approach to data analysis than structured usability tokens, allowing new insights to emerge from the data [20]. After being conducted, interview recordings were transcribed and recurring themes relating to the first research question were noted. All transcriptions were reviewed and for each participant addressing a topic relating to one of the themes, the participant's number was assigned to that theme. The second research question regarding method evaluation was subsequently assessed against the four adapted usability aspects from the cognitive dimensions framework.

Ethics

Prior to being introduced to the setup of the study, participants were asked to give their oral consent to participating. Participating in the study included being recorded during interviews and thinking-aloud sessions and having the data stored for analysis and documentation of results. Participants were also informed about the study's publication context and their role and anonymity in the study. All participants have given their oral consent to participate in the study.

RESULTS

The results of this study present themes emerging during participant interviews regarding the definition and evaluation of design system usability from a developer perspective.

Participants

Participants of the study included five male developers from the collaborating company, Intunio. Their experience of working with the implementation of user interface design (front-end development) ranged from 3 months to 13 years, and two of the developers also had one to three years' experience working as software designers. None of the participants had previous experience of working with the tested design system, although all of them were familiar with the term and did have experience of working with other design systems, style guides or component libraries. All developers followed an agile methodology in their everyday work. Interviews were held in English and Swedish, and interview responses in Swedish were translated into English by the author of this paper.

Usability themes

To answer the first research question: *How can design systems be defined and developed to improve developer experience and usability?*, four themes connected to improving design system usability for developers emerged: 1. *effectiveness and effort*, 2. *autonomy*, 3. *flexibility and complexity* and lastly, 4. *documentation and component code*. Secondly, themes regarding the second research question: *How can design system usability be evaluated in terms of developer experience?*, were identified and analyzed based on the adapted usability aspects, *understandability*, *abstraction*, *reusability* and *learnability*.

Effectiveness and effort

Three out of five developers expressed thinking the given design was easier to implement with the design system code components than they had initially estimated (P1-3); they satisfied with being able to create a functional and aesthetically pleasing user interface in the short amount of time (approximately 2 hours) allocated for the task, with a rather low effort (P1-3, 5). For example, participant 1 described the task being less complicated than initially estimated: *"When I received the design, and I thought about how I would implement it, then it was easier to implement than what I had expected, and that was thanks to the design system"*. Participant 2 felt the results were better than what would have been expected, based on the effort put in: *"It looks nicer than what I did, like I didn't do so much, and the end result was pretty satisfying. And that process in and of itself is pretty satisfying"*.

Participants felt especially satisfied with having the validation of form and styling of input fields already implemented and ready to use, as they experienced these being the most complicated subtasks (P1-3, 5). Participant 2 for instance, expressed being relieved to avoid arduous tasks: *"Form validation is a pain. Getting icons into [an input field] at the right position is an absolute pain, and it was nice that those problems were solved already... Having that be done already is extremely satisfying"*.

Four respondents (P1-4) also mentioned the benefits of creating one set of reusable components and treating that as the single source of truth for implementation in order to avoid redundant work or unnecessary discussions. Participant 1 stated: *"You have sort of a central point that defines how everything should be ... and then the people building the different apps don't have to communicate with each other, but everything is already in the design system"*.

Autonomy

Although experiencing a sense of relief for receiving some issues solved and implemented, four participants also expressed not minding, or even preferring, to implement the components themselves rather than receiving them solved (P1, 2, 4, 5). These developers also believed the design system might suit other developers, who have other preferences or less experience, better than themselves (P1, 2, 4, 5). Participant 4, for example said: *"If I have to do a bit more work to connect things and put them together, I don't think that's particularly hard or difficult"*, and continues: *"If you haven't worked with this for very long, or not done a lot of this ... then you might think this is difficult and that it's a lot more convenient to just get it pre-made"*. Participant 5 also expressed how others might appeal more to the idea of using implemented components: *"I guess I'm pretty unique in thinking [the language used to style coded components] is fun, and I'm pretty good at it. Many people love these things to avoid having to do it themselves"*.

All developers expressed being very familiar with implementing design similarly to how the design system code components were set up, and some developers experienced that the design system made them less able to make controlled changes than if they would have set it up themselves (P2, 4). Participant 5 for instance, expressed being familiar with the code: *"It could very well have been me who wrote these components, so I know how it all works with reusing small components ... how to build them to be generic"*. Participant 2 explained feeling constrained without complete ownership of the components: *"When I'm writing my own [components], in order to center the button I would have edited the button component, ... but now ... I know that I'm not really supposed to touch the design system, because it's not really my code"*.

Especially when touching on the issue of getting design requirements not covered by the design system, all participants expressed concern regarding the handling of design cases differing from the design system. Three participants particularly mentioned feeling constrained by using a design system and not fully in control (P2-4). Participant 3 for example, described a case where the lack of ownership could lead to complicated situations with other stakeholders: *"It's convenient in one way, but it's also less convenient if a customer says like '... we want ..., something that [the design system] doesn't support', then you're all of a sudden 'what do I do now?'"*.

Three participants (P1, 4, 5) suggested involving developers in the process of updating the design system in order to solve this dilemma. Participant 1 explained how the process of developing and updating the design system should be responsive to its users: *"The design system also needs to develop and satisfy the needs of the developers ... You have to make sure that both developers and designers who use the design system have the ability to give feedback to those who are responsible for the design system"*.

Even though the design system would provide instructions on how to handle such cases, participants were uncertain about whether the design system should allow being overridden, as they felt uncomfortable deviating from the design system (P1-5). Two participants also pointed out not wanting the design system too flexible, allowing incorrect usage (P1, 3). Participant 3 described this *"If I were to go in and override a button here ... I don't know exactly how I would have done that ... If that's even something you want to push, I don't know if you want to move away from the Intunio-branding design"*. Two developers also requested instructions on how the design should scale across different screen-sizes (P1, 2). Participant 1 said: "*I was worried when I saw that if I resized the screen, that [the input fields] didn't scale up*".

The feeling of being constrained also concerned some participants, since they feared it might slow down or block new ideas and narrow down the solutions base to what is only available in the design system (P1, 2, 4). Participant 2 explained how design systems can block problem-solving:

> *A design system can describe a popup window and what developers will do is they will implement a popup window for every action that you do ... then, well, our popup window looks exactly like it should in our design system, but in reality a popup window is not the correct solution to this problem. And I think design systems mask that problem a little bit too much, because it's easy to feel that you have done right*.

Two developers pointed out that design systems can help them keep other stakeholders in the team happy by finishing on time or maintaining the designer's vision, but that they themselves might have to make compromises in their code or feel less creative (P2, 5). Participant 2 described balancing between his own preferences and the aspirations of other stakeholders: *"You have to strike a balance between keeping everybody happy, I mean you want to keep your codebase happy, you want to keep your manager happy who wants the feature done in time, but you also want to keep the designer happy because they designed it in a given way and you want to respect that"*. Participant 5 expressed accepting less autonomy when needing to speed up the development: "*I think it's a lot more fun to build [the components] myself, but it's not as efficient ... It kind of depends too, of course if you just want to get something done as quickly as possible it was nice to avoid everything*".

Flexibility and complexity

Four developers expressed wanting more flexibility or customization when using the design system (P2-5). Participant 5 said: *"It turned out exactly the way the design looks. Just used the components, didn't need to write any [styling] of my own basically. A disadvantage with that, though, was that some things are difficult to adjust yourself"*. Three developers claimed to accept having to abide by the design system's rules if the problems it solved were more complex, as the rewards were higher (P1-3). Participant 2 stated the following:

> *Centering my button was... I had to do that in one specific way, and the system did not allow me to do it in the way that I actually preferred. Then, the form also had to be done in a specific way, but that's much more acceptable because the reward I get for doing it in the form's way is much higher because I get all of the other verification for free.*

Three participants also pointed out that the definition of complexity is different from developer to developer (P2-4). Participant 4 said: *"It is also a matter of definition, because what I think is complex, other people ... might not find complex and the other way around"*. One of the developers (P2) expressed that the difficulty with using a design system as a developer is that the level of complexity is static and not customized to your competence level: *"it's always a trade-off where you position yourself on a level between complexity and freedom ... if somebody asks you to ... implement this code with this design system, ... somebody else makes the decision for you"* and continued: *"I think it's much more of a human problem than a technical problem, which level of complexity you want in your design system is depending on which humans will use it"*. Two participants (P1, 4) also claimed that the issue of a design system is never the design system itself, but how it is implemented and used within a context. Participant 4 explained this: *"the design system as a system, I don't think, can lead to anything negative, but other things could be affecting.*

And that if it doesn't fit the project ... it's not the design system's fault that it doesn't fit ... it's probably rather that you're using the wrong design system".

Documentation and component code

When evaluating the documentation which described how to use the design system, all developers stated that they generally do not read documentation thoroughly or not at all, until receiving an error (P1-5), and that this was their approach when solving the implementation task as well. Participant 4, for instance explained: *"I didn't read the documentation at all until it went wrong ... I think it's easier to just do something first and if it goes wrong you can adjust that, most times it's not completely wrong"*.

Participant 1 explained that his reluctance to read documentation thoroughly was based on previous experiences of using documentation with too much information, resulting in wasted time and relevant information going missing: "*that might also make you skip reading texts because you know ... that effort you put into reading a text, it's not necessarily worth that effort because you don't know beforehand if it's going to be relevant*". Three participants (P1-3) pointed out that the design system should only contain the most relevant parts and not be bloated with edge cases, as that could also contribute to "obfuscation by information".

Regarding using pre-coded components, some participants (P2, 4) pointed out the importance of using code that stays true to the language they are written in and that using the components to write code should not feel any different from other code you write in that ecosystem. Participant 4 said: *"The design system shouldn't make you have to write code in a specific way, but it should rather work like the language you're using"*.

Design system usability evaluation

Besides providing general insights into developers' experience of using design systems, the second interview aimed at examining the methods used to evaluate a specific design system based on the four usability aspects: *understandability*, *abstraction*, *reusability* and *learnability*.

Understandability

Answers to the questions regarding understandability, i.e., the effort required to understand the semantics of the system based on names and documentation, showed that participants found the design system matched the design scenario very well, almost to the point that were concerned whether the design system would scale to other cases (P1, 2, 4, 5). Participant 2 said: *"all the information was there ... but my question is whether this would scale to a different example"*. Two participants also mentioned being concerned about the design system not scaling across screen sizes when addressing the abstraction aspect (P1, 2). Participant 1 said: *"I had to make a decision on how [the logotype] should work on different screen sizes"*.

Abstraction

Regarding the abstraction aspect i.e. whether the design system's abstraction level caters to usability, three participants requested the design system should provide them with clearer instructions how to override cases not matching the design scenario (P2, 3, 5). For example, participant 5 said:*"If you're going to handle special cases, that might be a bit difficult, perhaps"*.

Moreover, in relation to the abstraction aspect, the findings showed that four participants (P1-3, 5) requested more detailed information about how to position the components on the screen. Participant 1 also expressed wanting more information about positioning elements on the screen when addressing the understandability aspect: "*the design system didn't cover everything I wanted to do. I would have wanted to know more exactly where to place the logo and the design system didn't specify that*". Moreover, participants were not satisfied with having to wrap the components in more code in order to position the components or add extra styling to them (P2, 4, 5). This was discussed by one participant when addressing the reusability aspect, as he thought it added unnecessary code (P3). Participant 2 expressed preferring to do things differently if he was writing production code: "*I would have done things differently, like for instance I wouldn't have been satisfied with adding an extra [content division] element to center my button*".

Reusability

Although the adapted method focused mainly on identifying usability issues in the tested system, results from interviews also included aspects of using the design system the participants were content with. The questions regarding reusability, i.e. whether the system facilitates reuse and conciseness, showed all participants agreed the amount of code required to implement the design was about right (P1-5), enough to prevent the codebase from bloating without compromising the readability of the code (P2-5). Participant 4 said *"you can do these things ... so that it produces very very little amount of code, but completely incomprehensible, and that's not good. No, I think this was just about right"*.

However, three participants pointed out that the test case was too narrow or not complex enough to decide whether it facilitated reuse of their own code (P2, 4 ,5). Participant 2 said: "*inside my own code there wasn't enough kind of complexity in the app to reuse my own code"*. Two participants also mentioned the design system not covering enough use cases to be used in a real project (P2, 3). Participant 3 said: "*for a larger project ... I think ... this had perhaps covered 5 percent of the entirety*".

Learnability

When addressing the learnability aspect, i.e. whether the design system usage can be learned easily and incrementally, four participants declared learning rather quickly how to use the components by reading documentation, breaking it down into smaller tasks, starting with what they assumed they understood, and trying as they went on (P1-3, 5). Participant 1 for instance expressed this: "*When I had used one component, I was able to ... in the same way, use other components and also find documentation for those components easier*". Participant 4 felt the test case was too narrow to know whether the design system facilitated learnability: "*It's a pretty short ... example*

... to properly feel that effect, it's not that many tasks you're asked to do, so they can be done pretty quickly".

DISCUSSION

This section will discuss implications from the findings presented in the results section and bring forward topics of interest. The discussion is divided into two parts based on the two research questions.

Implications for improved design system usability

Autonomy and flexibility

Results from the semi-structured interviews showed autonomy and flexibility were key aspects when addressing design system usability from a developer perspective. Autonomy and flexibility, such as ownership of code and facilitation of customizability, were not only recurrently mentioned by developers as important factors throughout the interviews, but also influenced developers' opinions when discussing other themes. The importance of autonomy and flexibility within DX corresponds to previous research results showing that developers appreciate using systems that make them feel creative or professional, rather than the system being creative or professional [23] [25].

The findings of this study showed participants were not satisfied when the design system 'forced' them to solve tasks in one particular manner, and when having to deal with cases diverging from the design system. For this aspect, the participants expressed a need to feel ownership of the code components, in coherence with previous findings which shows that sense of control is a significant predictor of DX [24]. In order to incorporate ownership into the design system, participants suggested involving developers in the development process and considering them users of the design system. The notion of developers as users of development tools was explored by Kuusinen et al., who argued that an increased understanding of developers' dualistic role of being both producers and users of software can assist the improvement of project environments [23]. Correspondingly, participants' concerns that the tested design system could limit their autonomy and flexibility might also have been a contributing factor to why all developers felt the system might suit other developers better than themselves. Moreover, the findings illustrated the participants were willing to compromise their autonomy and flexibility only if the design system helped them avoid difficult, complex or boring tasks or to avoid redundant work or redundant communication with other team members. However, as pointed out by the participants, preferences regarding the demarcation of such tasks is highly individual which calls for compliance and customizability in the process of systemizing design and code.

Context of use

The development process or the context of use of the design system was also found to be a crucial factor when evaluating the usability of the design system by participants, who recurrently pointed out their concern with whether the design system would scale across different projects with different scopes. These findings indicate that there is an inherent difficulty of evaluating design systems separate from the process or the context of which it is implemented and used. Two participants explicitly acknowledged that the implementation context plays a fundamental part in evaluating the success of the implementation of a design system in a project case, rather than the design of the design system itself.

In turn, the contextual aspect of design system usability correlates to fundamental aspects of systems thinking, mentioned earlier as an instrument to approach wicked or fuzzy dilemmas, subsequently adapted by SOD to address complex challenges by designers. As emphasized by one researcher within the area of SOD, the scope of projects within software design and service design, such as design systems, is often limited and "the framing of the projects in these cases tends to be set by commercial interests only or conventions or directed by best practices instead of through active inquiry" [33, p. 3]. This researcher also suggests systemizing design should focus less on the resulting object and more on the complexity emerging from the interrelation of objects on a system level, in order to reach solutions that combine technological, social and commercial considerations for each individual case. Similar critique was communicated by some participants when talking about how they thought design systems could narrow down the solutions base within the software development process if the design system did not respond to its context of use. The notion of focusing less on best practices and conventions, and instead letting the fuzziness of each case determine how a shared set of practices and patterns can be approached to avoid wicked dilemmas within the process also implies that there in an inherent difficulty of using a service for documenting design systems, such as the one used in this study. As the customizability of such services are limited, the structure of all design systems documented using the service will rely on such conventions or best practices, rather than corresponding to the fuzziness of each particular case. This is perhaps a contributing aspect to why some participants had a hard time evaluating the appropriateness of the design system in regards to the scope and the scalability, because its structure was determined by something other than the implementation case, as the context of use was not properly incorporated in the test.

Documentation

Results also showed how participants approached the documentation in the design system, which was not predicted by the usability aspects. This showed participants generally do not read the documentation thoroughly, but rather try the code out, and then go back and fix things that do not work, and that this was also the case when using the tested design system. Similar to what participants expressed about using a design system in general, they pointed out that they were more likely to read documentation if it rewarded them by helping them solve complex tasks such as handling an error. This implies that participants generally approach the documentation in the design system autonomously, absorbing the information needed to solve their task through the documentation, rather than being guided by it.

Design system usability evaluation method

Based on the findings, the biggest drawback of the used method is that it is not extensive enough to evaluate all four adapted usability aspects properly, which some participants

mentioned regarding both reusability and learnability of the design system. Moreover, it showed the method did not provide answers strictly connected to the usability aspects, as some issues were found when discussing other aspects and some themes emerged which were not predicted to be found by the usability aspects at all. The evaluation against the four usability aspects did, however, find usability issues in the tested design system, such as the scaling of the case and the screen sizes. Furthermore, the thematic analysis provided findings connected to design system usability from a developer perspective such as the importance and individuality of autonomy and flexibility and how participants approached documentation of design and code in combination. The evaluation of the tested design system also highlighted the importance of contextual factors, as mentioned earlier, such as the working processes, the project to apply the design system on and the people using the design system. Future research could study developer experience and usability on real-life cases in existing projects to consider important contextual variables in the study. Usability perspectives from other users of design systems, such as designers and managers, could also be explored in future work.

To summarize, design systems have a multitude of applications within project cases and organizations, making the evaluation of such a tool extensive, complex and fuzzy. Moreover, this study was limited to evaluating usability aspects of design systems from a developer perspective. The findings were also limited to the perspectives of the 5 participating developers, and serves as an exploration of the topic of design system usability and DX. The results of this study highlighted the difficulties and the lessons to be learned from approaching such a system as a development tool, and evaluating it with regards to its DX. The results implicated that rather than considering a design system as a deliverable object for evaluation, the benefits of using a design system within software development projects reaches its full potential when applied onto an individual project case as a set of practices and patterns to approach fuzzy dilemmas emerging within that particular case.

CONCLUSION

This paper examined software development collaboration through the theoretical lenses of DX and Systems Oriented Design and presented a qualitative study on design system usability from a developer perspective. It adapted a method designed to assess API usability by endorsing participants to solve a programming task using the design system and following up with semi-structured interview questions based on the cognitive dimensions framework. The study was conducted on five developers, and results included participants' perceived usability issues emerging from the semi-structured interviews regarding using the design system in the programming test. The contribution to the field includes a deepened understanding of design system usability and DX in general, and a method for evaluating design system usability for developers in particular. Findings showed autonomy and flexibility were important factors when addressing design system usability from a developer perspective. The importance of context of use when evaluating design system usability was also discussed, which subsequently implied an inherent difficulty in relying on best practices for design systems, rather than creating patterns and practices to approach dilemmas emerging within each unique project case.

ACKNOWLEDGMENTS

I would like to give special thanks to all participants of this study who contributed vastly to this work with their thoughts and opinions. I would also like to thank Johan Frej and the people at Intunio for valuable input and support. Lastly, I would like to thank my supervisor Olga Viberg for useful guidance and feedback throughout the process.

REFERENCES

[1] ISO 9241-11:1998. 1998. *Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11: Guidance on usability.* Standard. International Organization for Standardization, Geneva, Switzerland.

[2] ISO 9241-210:2010. 2010. *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems.* Standard. International Organization for Standardization, Geneva, Switzerland.

[3] Christopher Alexander. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press. Google-Books-ID: mW7RCwAAQBAJ.

[4] Alisa Ananjeva, John Stouby Persson, and Anders Bruun. 2020. Integrating UX work with agile development through user stories: An action research study in a small software company. *Journal of Systems and Software* 170 (Dec. 2020), 110785. DOI: http://dx.doi.org/10.1016/j.jss.2020.110785

[5] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, and others. 2001. Manifesto for agile software development. (2001).

[6] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, and R. M. Young. 2001. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *Cognitive Technology: Instruments of Mind (Lecture Notes in Computer Science)*, Meurig Beynon, Chrystopher L. Nehaniv, and Kerstin Dautenhahn (Eds.). Springer, Berlin, Heidelberg, 325–341. DOI: http://dx.doi.org/10.1007/3-540-44617-6_31

[7] Johan Kaj Blomkvist, Johan Persson, and Johan Åberg. 2015. Communication through Boundary Objects in Distributed Agile Teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1875–1884. DOI: http://dx.doi.org/10.1145/2702123.2702366

[8] Ted Boren and Judith Ramey. 2000. Thinking aloud: reconciling theory and practice. *IEEE Transactions on Professional Communication* 43, 3 (Sept. 2000), 261–278. DOI:http://dx.doi.org/10.1109/47.867942

[9] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. DOI: http://dx.doi.org/10.1191/1478088706qp063oa

[10] Manuel Brhel, Hendrik Meth, Alexander Maedche, and Karl Werder. 2015. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology* 61 (May 2015), 163–181. DOI: http://dx.doi.org/10.1016/j.infsof.2015.01.004

[11] R. Buchanan. 2019. Systems Thinking and Design Thinking: The Search for Principles in the World We Are Making. (2019). DOI: http://dx.doi.org/10.1016/J.SHEJI.2019.04.001

[12] Steven Clarke. 2005. Describing and measuring API usability with the cognitive dimensions. In *Cognitive Dimensions of Notations 10th Anniversary Workshop*. Citeseer, 131.

[13] Jason Dagit, Joseph Lawrance, Christoph Neumann, Margaret Burnett, Ronald Metoyer, and Sam Adams. 2006. Using cognitive dimensions: Advice from the trenches. *Journal of Visual Languages & Computing* 17, 4 (Aug. 2006), 302–327. DOI: http://dx.doi.org/10.1016/j.jvlc.2006.04.006

[14] Thomas Erickson. 2000. *Lingua Francas* for design: sacred places and pattern languages. In *Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '00)*. Association for Computing Machinery, New York, NY, USA, 357–368. DOI: http://dx.doi.org/10.1145/347642.347794

[15] Fabian Fagerholm. 2015. Software Developer Experience : Case Studies in Lean-Agile and Open Source Environments. (Dec. 2015). http://hdl.handle.net/10138/158080

[16] Fabian Fagerholm and Jürgen Münch. 2012. Developer experience: Concept and definition. In *2012 International Conference on Software and System Process (ICSSP)*. 73–77. DOI: http://dx.doi.org/10.1109/ICSSP.2012.6225984

[17] Jennifer Ferreira. 2012. Agile Development and UX Design: Towards Understanding Work Cultures to Support Integration. In *Advanced Information Systems Engineering Workshops (Lecture Notes in Business Information Processing)*, Marko Bajec and Johann Eder (Eds.). Springer, Berlin, Heidelberg, 608–615. DOI: http://dx.doi.org/10.1007/978-3-642-31069-0_51

[18] Jennifer Ferreira, Helen Sharp, and Hugh Robinson. 2011. User experience design and agile development: managing cooperation through articulation work. *Software: Practice and Experience* 41, 9 (2011), 963–974. DOI:http://dx.doi.org/10.1002/spe.1012

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In *ECOOP' 93 — Object-Oriented Programming (Lecture Notes in Computer Science)*, Oscar M. Nierstrasz (Ed.). Springer, Berlin, Heidelberg, 406–431. DOI: http://dx.doi.org/10.1007/3-540-47910-4_21

[20] Hsiu-Fang Hsieh and Sarah E. Shannon. 2005. Three approaches to qualitative content analysis. *Qualitative Health Research* 15, 9 (Nov. 2005), 1277–1288. DOI: http://dx.doi.org/10.1177/1049732305276687

[21] Alla Kholmatova. 2017. *Design Systems: A Practical Guide to Creating Design Languages for Digital Products*. Smashing Media AG.

[22] Aïcha Konaté. 2018. Design systems at work: Optimizing design processes and aligning design work to company identity. (2018). https://aaltodoc.aalto.fi:443/handle/123456789/41399

[23] Kati Kuusinen. 2016. Are Software Developers Just Users of Development Tools? Assessing Developer Experience of a Graphical User Interface Designer. In *Human-Centered and Error-Resilient Systems Development (Lecture Notes in Computer Science)*, Cristian Bogdan, Jan Gulliksen, Stefan Sauer, Peter Forbrig, Marco Winckler, Chris Johnson, Philippe Palanque, Regina Bernhaupt, and Filip Kis (Eds.). Springer International Publishing, Cham, 215–233. DOI: http://dx.doi.org/10.1007/978-3-319-44902-9_14

[24] Kati Kuusinen, Helen Petrie, Fabian Fagerholm, and Tommi Mikkonen. 2016a. Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. 104–117. DOI: http://dx.doi.org/10.1007/978-3-319-33515-5_9

[25] Kati Kuusinen, Heli Väätäjä, Tommi Mikkonen, and Kaisa Väänänen. 2016b. Towards Understanding How Agile Teams Predict User Experience. In *Integrating User-Centred Design in Agile Development*, Gilbert Cockton, Marta Lárusdóttir, Peggy Gregory, and Åsa Cajander (Eds.). Springer International Publishing, Cham, 163–189. DOI: http://dx.doi.org/10.1007/978-3-319-32165-3_7

[26] M.J. Mahemoff and L.J. Johnston. 1998. Principles for a usability-oriented pattern language. In *Proceedings 1998 Australasian Computer Human Interaction Conference. OzCHI'98 (Cat. No.98EX234)*. 132–139. DOI: http://dx.doi.org/10.1109/OZCHI.1998.732206

[27] Nolwenn Maudet, Germán Leiva, Michel Beaudouin-Lafon, and Wendy Mackay. 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 (CSCW '17)*. Association for Computing Machinery, New York, NY, USA, 630–641. DOI: http://dx.doi.org/10.1145/2998181.2998190

[28] Marco Piccioni, Carlo A. Furia, and Bertrand Meyer. 2013. An Empirical Study of API Usability. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 5–14. DOI: http://dx.doi.org/10.1109/ESEM.2013.14 ISSN: 1949-3789.

[29] Horst W. J. Rittel and Melvin M. Webber. 1973. Dilemmas in a general theory of planning. *Policy Sciences* 4, 2 (June 1973), 155–169. DOI: http://dx.doi.org/10.1007/BF01405730 Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 2 Publisher: Kluwer Academic Publishers.

[30] Miika Ruissalo. 2018. Operating a design system in a large software company. (June 2018). https://aaltodoc.aalto.fi:443/handle/123456789/32488

[31] Cristian Rusu, Virginica Rusu, Silvana Roncagliolo, and Carina González González. 2015. Usability and User Experience: What Should We Care About? *International Journal of Information Technologies and Systems Approach (IJITSA)* 8 (July 2015), 1–12. DOI: http://dx.doi.org/10.4018/IJITSA.2015070101

[32] Dina Salah, Richard F. Paige, and Paul Cairns. 2014. A systematic literature review for agile development processes and user centred design integration. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. Association for Computing Machinery, New York, NY, USA, 1–10. DOI: http://dx.doi.org/10.1145/2601248.2601276

[33] Birger Sevaldson. 2013. Systems Oriented Design: The emergence and development of a designerly approach to address complexity. In *DRS Cumulus Oslo 2013 - Proceedings of the 2nd International Conference for Design Education Researchers (14-17 May)*, Vol. 4. ABM-media, Oslo, Norway. https://dl.designresearchsociety.org/conference-volumes/39

[34] Susan Leigh Star and James R. Griesemer. 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science* 19, 3 (1989), 387–420. https://www.jstor.org/stable/285080 Publisher: Sage Publications, Ltd.

[35] Jenifer Tidwell. 1997. Common ground: a pattern language for human-computer interface design. (1997). http://www.mit.edu/~jtidwell/common_ground.html

TRITA-EECS-EX-2021:117